BioCompute:

Harnessing Distributed Systems for Bioinformatics Applications

Patrick Braga-Henebry

May 2009

1 Abstract

In recent years, the size of bioinformatics datasets has followed an exponential growth trend which show no signs of abating. While established genome search algorithms exist, specifically BLAST, an algorithm which identifies similarities between a genomic sequence and a database of other sequences, the growth of computer storage and datasets has outstripped the growth in processing power.

Analysis of the problem domain and performance measurements indicate that a distributed computing model can provide significant and scalable performance benefits to BLAST. Genome search is data-intensive, and sequence databases frequently range in the gigabytes; established non-data-intensive distributed computing models do not suffice.

The aim of the project is threefold:

1. to build a framework for executing batch bioinformatics jobs using existing distributed computing resources, specifically the Condor pool at Notre Dame;
2. to provide a user-friendly tool for bioinformatics researchers to process BLAST search queries;
3. to provide users a useful and unobtrusive record of queries they conducted.

A system called BioCompute was developed atop Condor to meet these goals. Condor is a distributed batch computing system used by many institutions around the world. Notre Dame's Condor pool comprises several hundred machines across campus.

Using BLAST as a reference application for the first goal, a model was developed in which input queries were distributed across a static pool of hosts pre-seeded with multi-gigabyte datasets. Currently, BioCompute runs on 32 nodes. The interface and framework were constructed with modularity in mind in order to facilitate adding new applications in the future. To satisfy the second goal, a web interface was developed to submit, monitor, view, and explain BLAST jobs and results. This interface should accommodate users with a wide range of familiarity with the original BLAST tool. In satisfaction of the third goal, this interface records the parameters and details of each

job submission to ease reproducing results and provide a paper trail of conducted queries and comments on the query results.

The functional goal of this tool is to empower biologists to conduct large BLAST searches in a smaller time frame by hiding the implementation details of the underlying computing model.

2 Acknowledgments

I would like to thank Dr. Douglas Thain for his guidance and instruction over the past three semesters, and for introducing me to interesting problems and methods for approaching them.

Many thanks also to Dr. Scott Emrich, for his guidance and input on BioCompute, and to Rory Carmichael, for his contributions to the project, and for agreeing to put up with my code after I graduate.

In addition, thanks to Dr. Jeanne Romero-Severson and all the other beta-testers for BioCompute, whose suggestions and insights continue to help us iron out the inevitable kinks.

Thanks go to C.V. and A.T. for their companionship during late-night thesis-writing sessions. Finally, thanks to all my friends and family for their support and fellowship throughout my four years at Notre Dame.

Contents

List of Figures

List of Tables

3 Introduction

3.1 Growth Trends in Bioinformatics

The size of bioinformatics datasets has grown exponentially in recent years, and shows no signs of abating. With increases in sequencing speed and parallel efforts being conducted across the globe on many genomes, an explosion of genomic data is occurring. Figure 1 shows the exponential increase in base pairs and sequences stored in NCBI's GenBank repository since 1982.

While the collective speed of sequencing and annotating sequences has increased—yielding larger and faster-growing datasets—the resulting volume of data is problematic from a data management and processing viewpoint. The cost and availability of computing power has roughly followed Moore's Law, which postulates that the number of transistors on a chip will roughly double every eighteen months, this trend has slowed or been transformed in recent years by the advent of multi-core processors. Additionally, other bottlenecks to efficient processing of this data include the I/O speeds of disk drives, which have not scaled in proportion to increasing disk size and decreasing cost.

3.2 BLAST and Condor

Distributed computing systems can alleviate this growing mismatch between available computing power and the growing volume of bioinformatics data. Distributed systems have been developed to effectively harness and utilize the computing power of many connected systems and the parallel storage of multiple attached local disks. Condor [11] is one such system which was developed specifically to harness idle workstation computing power to provide collective distributed computing resources to the community.

BLAST [2] is a widely-used data-intensive bioinformatics application. It accepts nucleotide or peptide sequences as input, conducts alignment searches against much larger sequence databases, and returns a ranked set of matching sequences. Multi-sequence

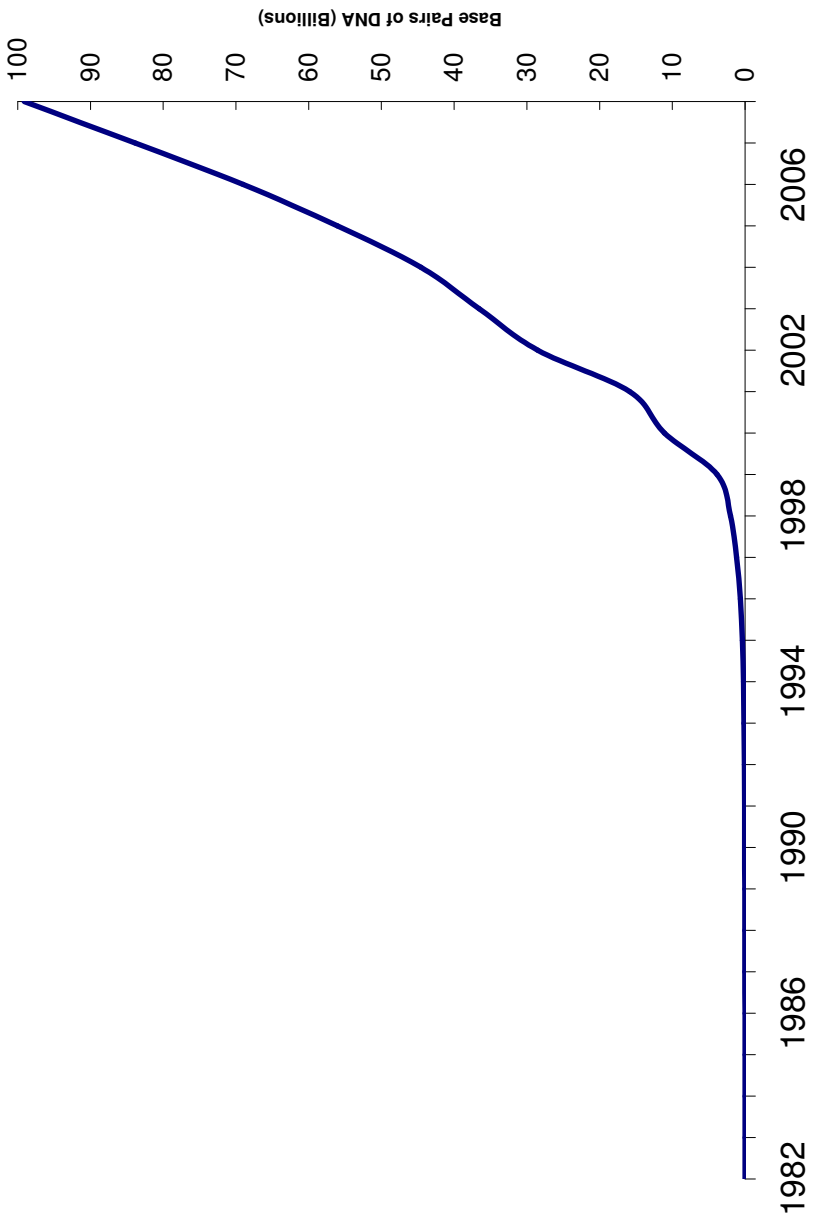

Figure 1: Growth of GenBank in recent years [6].

BLAST queries are trivially parallel and thus are easily distributable.

There are many practical and logistical barriers intrinsic to harnessing distributed computing resources. Reliability and fault-tolerance are concerns when dealing with multiple independent systems. Effective division of the processing task is another, along with load-balancing and dealing with heterogeneous distributed execution environments. In short, harnessing distributed resources, while possessing the potential for high return on investment, is complicated and difficult.

3.3 Providing a usable BLAST abstraction atop Condor

Such systems present intrinsic usability difficulties even for those versed in their construction and use. The necessity of such systems will grow and spread as the volume of data grows in many domains. To facilitate growth, special attention should be paid to the usability and interfaces of such systems to make them approachable and understandable by those not versed in distributed computing but in the relevant problem domain.

BioCompute aims to provide an abstraction of distributed computing resources for bioinformatics applications. By hiding details of the distributed system and providing a usable domain-pertinent interface, BioCompute can both improve bioinformatics runtimes and facilitate usage of distributed resources. It also aims to aid in data management by providing tools to annotate and detail experiments.

BioCompute has been developed to handle distribution of BLAST queries. Work is in progress for the construction of a modular, application-agnostic system for which modules can be written to provide distributed implementations for different applications.

This paper will discuss the implementation of BioCompute to distribute BLAST queries. Section 4 reviews related works, while sections 5 and 6 discuss the implementation of distributing BLAST via Condor and handling replication of BLAST sequence databases, respectively. Section 7 discusses performance and affecting factors, and section 8 addresses user-facing usability concerns.

4 Related Works

4.1 Distributed Systems

4.1.1 MapReduce

MapReduce [4] is a distributed programming model and computing system developed at Google for data-intensive applications. It is modeled and named after the map and reduce primitives in LISP: a map function is applied to each item in a list to produce a list of resulting items, and a reduce function is executed on lists of the resultant items to produce a scalar result. By restricting users to this functional programming model, MapReduce can use a common distributed execution model for a variety of real-world applications.

In their programs, MapReduce users specify a map function and a reduce function. The input data is split into pieces and each piece is fed to an instance of the map function, which emits values which are fed to instances of the reduce function. By forcing users to write their programs in a manner which emphasizes modular and independent computations, MapReduce can efficiently distribute execution of the program on thousands of clusters.

MapReduce seeks to make it easy for developers to harness distributed computing resources to execute very large data-intensive jobs while hiding logistical concerns typical to distributed computing, e.g. fault tolerance and load-balancing. It is implemented as a C++ library against which a user must write, link, and compile their programs.

While BioCompute's division input division strategy closely resembles that found in MapReduce, BioCompute would have to undergo many changes to be implemented on MapReduce. First, the binary BLAST application would have to be modified in some fashion in order to hook into the MapReduce structure, where no such modification has taken place in the current implementation. Also, running BLAST jobs require querying large sequence database files which are not trivially divided nor distributed. MapReduce's

programming model does not accommodate such large, static resources, so the database would have to be divided among jobs, which would introduce domain-level algorithmic complications, and further modifications to the binary application. In short, BioCompute as-is could not run on MapReduce, but similar computations could be accomplished after non-trivial modifications to the execution framework.

4.1.2 Condor

Condor [11]is a distributed computing system developed at the University of Wisconsin-Madison. Based on the fact that many personal workstations possessed on-demand computing power which often sat unused, it was originally developed as a tool to pool and harness idle workstation CPU cycles. As such, it is built to handle highly heterogeneous Unix cluster environments. It was built to respect the rights of the machine owners, who can specify the level of Condor usage on their machines, and remove a Condor job from their machine at any time.

Jobs submitted for execution in the Condor pool are described by a Condor submit file. This file details the inputs, executable, outputs, and parameters of the job and requirements of the job environment. Condor has a classification system called "ClassAds", which describe attributes of a machine's execution environment in name-value pairs. When jobs are submitted to the Condor system, attribute values are specified, and machines which match the specifications are considered for job execution.

Condor is a flexible environment. Programs typically do not require recompilation or relinking to run in the Condor environment, nor are they constrained to a particular programming model or language. Unless otherwise specified, Condor filters nodes using ClassAds to find an execution environment matching the submitting computer (i.e. 32- vs. 64-bit, hardware architecture, etc).

Reliability and fault-tolerance is a major concern in distributed systems. In the event of an expected shutdown of a machine running a Condor job, Condor can package up the

execution and resume the execution on another machine. In the event of an unexpected shutdown, e.g. someone accidentally unplugging a machine, Condor can restart the job on another eligible node.

There is a Condor pool at the University of Notre Dame composed of hundreds of machines of different varieties. Computer lab workstations, faculty workstations, and various computing clusters are included. Condor is the underlying distributed system upon which BioCompute relies for distributed execution.

4.2 The Chirp Filesystem

Alongside Condor, Notre Dame also runs a distributed file-system named Chirp [10]. The file-system was designed to facilitate data-transfer within local-network computational grids.

Chirp aims to provide unprivileged deployment, simple interfaces, familiar access controls, and flexibility to accommodate different types of use cases. Use of Chirp is commonly facilitated through Parrot [9], an *interposition agent* which presents files on various nodes as a directory tree in a Unix system and accordingly rewrites system calls to those resources.

Chirp has two main parts, the *chirp_server* and the catalog server. The *chirp_server* is the client which operates in user-space on a node, provides files for distributed access, and periodically informs the central catalog server of its presence and status. The catalog server maintains and reports information about all the Chirp nodes. By maintaining a periodically-updated central list and caching it on nodes, Chirp can provide fast and local inquiries into the top-level of the file-system.

BioCompute employs Chirp alongside Condor to aid in efficient and simple batch distribution of BLAST sequence databases across the Notre Dame network. Chirp provides a simple interface for a peer-to-peer file-distribution mechanism which reduces total distribution time.

4.3 Distributing BLAST

4.3.1 University of Iowa BLAST Cluster

Braun et al. detail three possible approaches to distributing BLAST queries among a cluster and discuss the implications of the design options they made in designing a cluster for batch processing of BLAST queries [3]. The first and most fine-grained approach is to parallelize to the level of comparing two individual sequences against one another, one source and one target. The next approach to dividing BLAST queries they mention is to partition the genomic databases into "chunks" and distribute these chunks among several nodes. When a query is submitted for processing against a database, the query is submitted against all chunks of the database, and the results are returned and merged. The third and most coarsely-grained approach entails storing complete databases on nodes and splitting up the set of incoming query sequences across many nodes, each with a complete copy of the database against which to compare results.

At the time of the writing of the paper, only the coarse-grained parallelization of BLAST queries had occurred, and work was in progress on the medium-grained approach involving partitioning the databases.

The cluster accepted job submissions from two sources: a daily batch input from an array of sequencing machines and a web interface. At the time of writing, the vast majority of system use came from the batch processing. The researchers noted that at the time of writing, 90% of use cases involved querying a single sequence against a database. In this case, coarse-grained distribution of BLAST will not result in any performance improvement of processing because there would only be one sequence to run against one database.

The paper noted that an implementation of the medium-grained database distribution method was nearing completion. Since the sequence datasets are divided in this implementation, the merging of results from each sequence-database comparison in order

to maintain output-compatibility with a normal, sequential BLAST is non-trivial. Once all jobs have completed and returned results, the results must be sorted. The E-values, the number of significant alignments expected by chance, must be recomputed to reflect the actual size of the whole database rather than the size of each individual partition. Also, the results must be sorted by each match's score, which is a function of the length of the sequence alignment.

This coarse-grained implementation described in the paper closely resembles BioCompute's division strategy. BioCompute's only source of input sequences is through a web-based interface, but the character of input is different than what the authors expect in their system. Queries sourced from the web interface in their system are treated as time-critical and have operational preference over batch jobs. In BioCompute, web-based jobs are treated as batch queries and are expected to have a potentially lengthy runtime. While there is emphasis on immediate feedback to the user in BioCompute and the system is built to deliver processing time improvements to BLAST, the user interface does not expect nor encourage the user to expect immediate results processing.

It should also be noted here the BioCompute's BLAST distribution strategy assumes sequence input files will multiple queries, often many thousands of queries. This has been the observed standard practice of bioinformatics researchers at Notre Dame.

At full capacity, the sequencing systems feeding the batch processes produce 2880 sequences per day. Typically eighty percent of the sequence pass verification steps and are fed to BLAST. The paper reports that a daily load of 2310 sequences are processed by the cluster in 20 hours, keeping up with the daily production of sequences by the sequencers. By contrast, a single node at the time would have taken more than three weeks to process the same dataset.

4.4 Web Portal

4.4.1 RIKEN Bio Portal

The Advanced Center for Computing and Communication at the RIKEN Institute in Japan saw a need for a user-friendly system for life sciences researchers to access distributed computing resources at the Institute [12]. They noted that many life sciences researchers were unfamiliar or uncomfortable with traditional computer user interfaces, especially the command line. They developed Bio Portal, a web interface written in Java using the Commodity Grid Toolkit, to streamline and simplify using the RIKEN Super Combined Cluster (RSCC) in order to process BLAST and ClustalW jobs.

Using Bio Portal, life sciences researchers ("wet" researchers) are able to easily upload genomic or protein sequences for processing, view and download results, stop progress of currently-running jobs, and delete jobs. Also, frequently researchers would need to use results from BLAST as input for a ClustalW job, and the process for doing so manually using Bio Portal's system was slow and cumbersome, so a BLAST+ClustalW processing option was added which executed this chain of processes automatically.

The RSCC uses Hi-Per BLAST, a parallelized version of BLAST which maintains result compatibility. While large jobs were expected to effectively utilize the computing resources of the RSCC, it was postulated that smaller jobs would not benefit in runtime from the cluster's resources because of higher temporal overhead when using the RSCC. A separate server was added to Bio Portal to process jobs with fewer input sequences. After conducting trials, the team concluded that jobs against large databases such as "nt" should be run on the RSCC cluster, while jobs against smaller databases such as "sts" should be run on the separate server. They found that only when comparing against databases of intermediate size such as "patnt" should the number of sequences be using as a tool in estimating job runtime and selecting which computing resource to use. They found that database size is more influential in resource selection than the number of

sequences.

A separate group in RIKEN hosts mirrors of public databases, and Bio Portal synchronizes their databases daily with these mirrors and provides status indicators and timestamps for each database.

5 Distributing BLAST Jobs

5.1 The BLAST Application

5.1.1 Description

BLAST is a collection of algorithms which identify similarities between a genomic sequence and a database of other sequences. When BLAST is run, each query is compared against the specified database according to the chosen algorithm, and the algorithm returns a the names of sequences found in the database which are similar to those in the input file.

Because BLAST is only an application of BioCompute, the original BLAST algorithm itself is not modified. The BLAST executable is used in its original form, without any modifications or recompilations. The focus of BioCompute is to implement a generalized pattern of abstraction atop Condor allowing for multiple applications to be expressed and used by those with little or no background in formal computing.

5.1.2 Inputs

There are two input vectors for BLAST execution, hereafter referred to as input files and databases. BLAST input files are in the text-based FASTA format specified by the NCBI. They contain a list of queries, and are supplied by the application user. The beginning of each query is noted with a $\rangle$, followed by the a description of the sequence. The description typically includes a name and identifier number, ended with a newline. Sequence names and identification numbers often follow conventions set by the NCBI. A nucleotide or peptide sequence follows, representing base pairs, amino acids, or patterns thereof in a standardized text representation. Whitespace within the sequence is not significant, but lines shorter than 80 lines are recommended for readability. A simple example of FASTA-formatted file containing peptide sequences is shown in Figure 2, while Figure 3 shows a nucleotide sequence in FASTA format. Typical input files for

```
>1397save_transcriptional_regulator_GCN4
VPESSDPAALKRARNTEAAR
RSRARKLQRMKQLEDKVEEL
LSKNYHLENEVARLKKLVGE
R
>1381save_cobratoxin
LECHNQQSSQTPTTTGCSGG
ETNCYKKRWRDHRGYRTERG
CGCPSVKNGIIINCCTTDRC
NN
```

Figure 2: A sample peptide sequence in FASTA format

```
>MAGI_43307_1
AAAGTTGAGTGGTTGGTAACA
GTTACCGTTCTTAAGATTGAT
CAACTATGGTGGTACGAGTCT
TGTAGAAAGTGCCTCAAGAAA
ACAAAGCCTCATGGTGATGCC
TATAAATGCTCGGACTCTGGT
TGTGGCCATGTTGGTCCGCCT
AATCCAAGGTACAGGTTGCTC
ATCACAGCAGGAGATGAGACAG
```

Figure 3: A sample nucleotide sequence in FASTA format

BioCompute's initial target use cases range from several hundred bytes to dozens of megabytes.

BLAST databases are large compilations of sequences in a binary format. The databases used in BioCompute range in size from several megabytes to several gigabytes. Many databases are maintained and hosted by the NBCI including the standard non-redundant database *nr*. Table 1 shows a list of 39 sequence databases currently host by BioCompute, sourced from both the NCBI and datasets provided by researchers at Notre Dame.

Name	Size
Heliconius_e_unigenes	108K
ACWP2_contigs_v1	592K
ABWP1_contigs_v1	752K
CCMHS-1_contigs_v1	824K
CCWP1_contigs_v1	872K
ABWP2_contigs_v1	1.3M
ROB-2_contigs_v1	1.5M
WOA-2_contigs_v1	1.5M
ROA-2_contigs_v1	1.5M
WOB-1_contigs_v1	1.8M
Heliconius_e_proteins	2.1M
ACHS1n_contigs_v1	2.2M
ACWP1_contigs_v1	2.3M
WOB-2_contigs_v1	2.4M
WOA-1_contigs_v1	3.0M
ACCanker_contigs_v1	3.2M
ROA-1_contigs_v1	3.4M
CCWP2_contigs_v1	3.7M
ROB-1_contigs_v1	3.9M
PfalciparumAllTranscripts_PlasmoDB-5.5	4.1M
CCNHS-1_contigs_v1	5.1M
ACHS2n_contigs_v1	5.3M
CCCanker_contigs_v1	5.8M
1_celera_contigs	6.3M
2_celera_contigs	7.1M
bombyx_predicted_proteins	7.2M
CCMHS-2_contigs_v1	11M
CCNHS-2_contigs_v1	13M
ButterflyBase	18M
lepbase	22M
Fagacea_db.fasta	30M
Fagaceae_db	87M
uniprot_sprot	233M
Drosophila_melanogaster_unigenes	240M
agambiae.CHROMOSOMES-PEST.AgamP3	261M
pataa	356M
patnt	1.7G
uniprot_trembl	3.5G
uniprot_trembl.fasta	3.6G
nr	6.4G

Table 1: List of sequence databases hosted by BioCompute and sizes.

5.1.3 Outputs

The original, unmodified BLAST program has many multiple output formats which are not equivalent in form or content. All of these output options are presented to the BioCompute user. The output of a BioCompute BLAST job matches the output of an original BLAST in content, but the form may be slightly different; depending on the output type selected, additional program header information is scattered throughout the file, one for each of the distributed Condor jobs. This output file could be normalized and scrubbed to more closely represent the form of original BLAST queries, but this process would be complicated by the myriad output options. Figure 4 shows an example of the default output option for BLAST.

5.2 Comparison to MapReduce

A BLAST input file can have any number of query sequences. Typical jobs run during BioCompute's testing range from tens to hundreds of thousands of queries. Each query sequence is compared independently against the specified database and the results of queries are not dependent on the results of other queries. Because of their independence, these queries need not be compared against the database in any particular order, nor need they be executed on the same machine; the value of the function is determined by the input sequence, the blast algorithm, and the database.

In order to distribute the queries among nodes, the original input file is split into batches of N queries each. Each of these resultant input query shards now can be run against a BLAST database to produce the sequences matching those in the input file. This set of matching sequences is a partial result for the query of the original input file against the database, but it is a complete result for the query of the input shard against the database. These self-contained subsets of results are then combined into one result, corresponding to the result for the original input file. See Figure 5.

This model of dividing input and distributed execution closely resembles Google's

```
Reference: Altschul, Stephen F., Thomas L. Madden, Alejandro A. Schaffer,
Jinghui Zhang, Zheng Zhang, Webb Miller, and David J. Lipman (1997),
"Gapped BLAST and PSI-BLAST: a new generation of protein database search
programs",  Nucleic Acids Res. 25:3389-3402.

Query= Contig2
         (910 letters)

Database: All non-redundant GenBank CDS
translations+PDB+SwissProt+PIR+PRF excluding environmental samples
from WGS projects
           7,135,729 sequences; 2,462,332,163 total letters

Searching.....................................................done

                                                                  Score    E
Sequences producing significant alignments:                       (bits) Value

ref|YP_002221402.1| NADH dehydrogenase subunit 2 [Blastocystis s...    37   2.1
gb|ACD10933.1| NADH dehydrogenase subunit 2 [Blastocystis sp. Na...    37   2.1
ref|YP_001096002.1| NADH dehydrogenase subunit 4 [Metaseiulus oc...    37   2.8
ref|YP_001096013.1| NADH dehydrogenase subunit 4 [Metaseiulus oc...    37   2.8
ref|ZP_00239675.1| hypothetical protein protein [Bacillus cereus...    36   4.7

>ref|YP_002221402.1| NADH dehydrogenase subunit 2 [Blastocystis sp. NandII]
 gb|ACH86083.1| NADH dehydrogenase subunit 2 [Blastocystis sp. NandII]
          Length = 493

 Score = 37.0 bits (84), Expect = 2.1
 Identities = 24/81 (29%), Positives = 43/81 (53%), Gaps = 5/81 (6%)
 Frame = -1

Query: 829 FYHFKITFSLSDIDIVFDILLEIGENFIVYGLLYSALV----FLIMYYFVTYCVDNR-YI 665
           +++ KI + LSD+  V+     +G +FI+     + +     FL  +Y    C++N  Y+
Sbjct: 359 YFNNKIKY-LSDLSYVYKYNKTLGLSFIITMFSMAGVPPMAGFLAKFYSFFVCIENNFYL 417

Query: 664 LATLNDGLPKISNFYFVSFIK 602
           LAT+   L  I  FY++ F+K
Sbjct: 418 LATIGVLLSIICTFYYIRFLK 438
```

Figure 4: A snippet of a typical BLAST output in the default format.

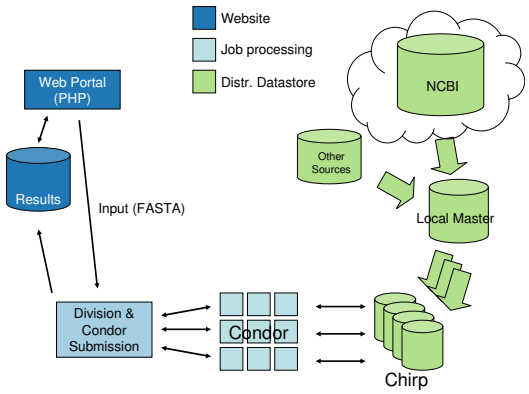

Figure 5: Outline of job execution

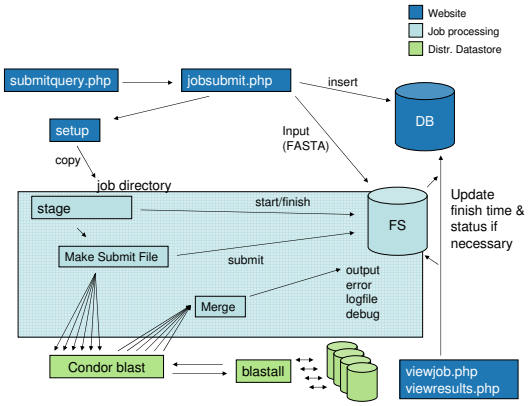

Figure 6: Data flow in BioCompute

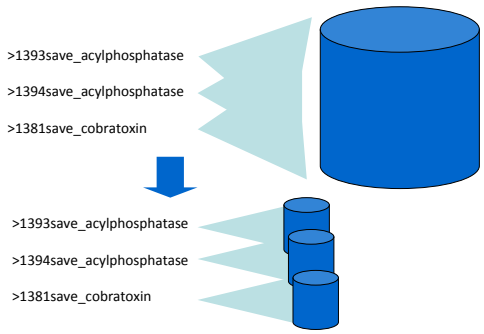

Figure 7: Divided input sequences run against a set of replicated databases

MapReduce model in some respects: a problem is divided into independent subproblems, a map function is evaluated on the input splits, and the intermediate values are reduced into a result for the original problem. In BioCompute, the input data is the FASTA sequence file originating in the user, the map function is the BLAST executable operating on an input shard and necessary databases, and the reduce step is simply an ordered concatenation of the output streams of the independent, distributed subproblems.

BioCompute differs from MapReduce in that it requires large, static sequence databases, which are treated as atomic units to reduce complexity in the problem domain. This is discussed more in Section 6.2. MapReduce does not handle large, indivisible files. In order to provide these resources in whole on remote execution nodes, BioCompute distributes these databases in batch ahead of time, and therefore can assume their existence on remote nodes when scheduling work processes.

5.3 Execution Overview

Each BLAST job submission to BioCompute is executed using the Notre Dame Condor pool. The following is the sequence of events which occurs for every job submission.

1. The user uses the "Submit Query" page on the web interface to select a local FASTA file to upload, choose a database to run the sequence against, specify BLAST options, and name the job. The user submits the page and an HTTP post is submitted to the submission processing page.

2. The submission processing page creates an entry in the database for the job, creates a new job directory on the server, and copies the sequence input file to said directory. The current set of job execution scripts is copied into the directory as well.

3. The job-execution script is called. It's first task is to split the sequence file into segments of N queries each. A Condor submit file specifying the number of jobs to run and their respective inputs is then generated. A sample submit file is shown

in Figure 8. At this point, the BioCompute job is ready for submission to Condor: there are K input files of at most N queries each, and the script which Condor will distribute is in the job directory and ready to be copied to BioCompute worker nodes.

4. The job is submitted to Condor. Once jobs are dispatched to BioCompute computing nodes, the script checks for the existence of the required database on the local file-system. If it is found, then the job will continue normally and BLAST will execute on the input split. When a job is complete, its standard output and error streams are sent back to the master job directory and stored in the indicated filenames.

5. Upon completion of all jobs, the results from each job split are concatenated in order. This is a linear time step, but if necessary in the future, this step can be easily divided as it is simply concatenation of ordered, numbered strings. After this merge is completed, a timestamped "completed" file is written to the directory indicating the job is complete. Optionally, an email can be sent to the user notifying them of the job's completion.

6. Upon a user requesting to view the job through BioCompute's web interface, the status of the job recorded in the database is checked. If the job is not in a terminal state indicating its status will not change, i.e. "Complete" or "Complete with Error", the existence of the "completed" file in the master job directory is checked. If it is there, errors in execution are checked for, the database is duly updated and the newly-updated status is displayed to the user.

5.4 Submit File Overview

The Condor submit file details execution details of a distributed job. Many parts of a submit file are frequently similar across many different types of jobs. This section will

review the salient and unique aspects of the sample submit file shown in Figure 8. This sample submit file is representative of those generated for BioCompute jobs in general.

The first differentiating part of the submit file is the requirements line. This line is specifying what BioCompute requires of a node in order to run. The first clause specifies the machine group. Currently, BioCompute nodes are restricted to the cluster designated *sc0* within the Condor pool. The next attribute restricts BioCompute jobs to only run on the first CPU of a machine; under the current configuration of the Condor pool, each CPU of a multi-cored machine can be assigned jobs independently. The final clause ensures jobs are only assigned to machines where memory is currently available. These last two clauses were added because it was found that BioCompute jobs would frequently deadlock when two jobs were scheduled to two CPUs on the same machine. Many of the sequence databases in BioCompute are larger than available memory and as a result the jobs are I/O-bound. Scheduling two processes on the same machine would result in detrimental competition for available memory and disk access.

The memory and CPU restrictions were added to reduce the possibility of this happening. This could result in reduced performance for jobs against databases which are smaller than half the available memory because otherwise jobs would be able to scheduled on both CPUs of a machine without negative effects. It should be emphasized here that the *sc0* cluster is a shared cluster and not exclusive to processing BioCompute jobs, and therefore there could be other processes running and consuming memory on a machine. The policy of restricting to a single CPU has performed suitably so far for databases of all sizes.

The next salient attribute of the submit file is the arguments line. This is the set of command-line parameters which will be fed to blast.script, a wrapper for the BLAST executable. The parameters include the command-line options which specify how the job will be executed. These include which variation of BLAST will be performed (e.g. *blastn*, *blastp*, *tblastx*), which database to compare against, the output format, and the

number of sequences and alignments to show.

In addition to the BLAST job parameters, the BioCompute-assigned ID number is passed along, and the Condor job number is indicated with the $(PROCESS) macro. The Condor job number indicates which division of the job is being executed and is in the range [0,Q), where Q is the number on the *queue* line of the submit file. The output and error files which are transfered back to the master job directory are marked with the Condor job ID to designate order and provide ensure unique filenames.

The *on_exit_remove* line directs Condor how to handle a job which fails to find the requisite database on the local node. When that happens, the script exits with error code 101, indicating to Condor that it should be removed from that local node and reassigned to another eligible node.

5.5 Remote Execution Script Overview

Three noteworthy events occur in the remote execution script on BioCompute nodes:

1. The script queries the local Chirp daemon for the local path of files stored on the node.

2. The script checks for the existence and accessibility of the database it will query against.

3. If the database is readable, the script calls the BLAST executable.

The sequence databases are stored locally on the BioCompute nodes through the Chirp distributed storage system. Access to the Chirp file-system is usually obtained through the *chirp_server* process, but in the special case where the requisite files stored via Chirp reside on the local node, as is the case in BioCompute, faster access can be obtained by accessing the local files directly and circumventing *chirp_server*. To do this, the script requests the local Chirp path from Chirp via the *localpath* directive and

```
universe = vanilla
executable = ./blast.script
requirements = (MachineGroup=="sc0") &&
               (VirtualMachineID==1) && (Memory > 0)
arguments = 194 blastn 0 ButterflyBase 194.split.$(PROCESS) $(PROCESS) 50
output = 194.output.$(PROCESS)
error = 194.error.$(PROCESS)
transfer_files = always
transfer_input_file =194.split.$(PROCESS)
on_exit_remove = (ExitCode!=101) || (ExitBySignal==TRUE)
log = 194.logfile
queue 2116
```

Figure 8: A typical Condor submit file for BioCompute jobs

proceeds to access the files through the local file-system. In addition, to allow this, the ACLs of the BioCompute database directory must be set to allow local access.

The second item of note is that the script checks for the existence of the database on the local node before calling the BLAST executable. If it is not found, the script exits with code 101, directing Condor to reschedule it to another node, as mentioned previously.

5.6 Reporting Job Status

A Condor job generates and maintains a logfile in the master submission directory. Upon job completion, BioCompute generates a file marking the time of completion. Using these two sources of information along with the size of the standard out stream, BioCompute currently reports a job as Submitted, Running, Complete, or Complete with Error. A new system is being built which will rely solely on the Condor logfile and give more granular status reporting.

5.7 System Reliability

BioCompute is entirely dependent on the underlying Condor system for job execution. Condor is a mature and proven distribution platform. BioCompute should thus "hand off" jobs to Condor for execution as quickly as possible in order to reduce the domain for errors and potential for job failure. By pushing responsibility to Condor quickly, BioCompute reduces the possible space for job failure.

6 Handling Sequence Databases

6.1 Distributing Databases

Distributing BLAST across a set of nodes differs from the MapReduce model in its dependency on large databases. In MapReduce, all inputs to the program are expressed as splits of a list, while the BLAST there is the additional dependency of a database. It is not feasible to simply distribute the databases on-demand per each instance because of their relatively large size and the relative paucity of network bandwidth, a frequent bottleneck in distributed systems.

Complete copies of the required databases are stored locally at each node using the Chirp distributed file-system. Databases are added by first staging the requisite files in a master copy which is not used for production job execution. To enable reproducibility of results, once a database is added to the system under a specific name and release date, it will not be updated. Rather, the updated version of the database would be distributed under a different release date.

Adding a new database to the system is handled by a set of scripts which first push the new database onto a single node in the cluster and then distributes the data among the cluster by making a call to the *chirp_distribute* function provided by the Chirp file-system. This function efficiently distributes a file or directory on one node to multiple specified Chirp nodes.

There may be a significant portion of time during which a newly-added database will be locally stored on a fraction of the BioCompute nodes. Upon arriving at a node for execution, each BioCompute job checks for the presence of its required database. If it is not stored on the machine, the job exits with an exit code specifying that the database could not be found on the machine. The Condor job handler will then reschedule the job at another node. In this way, new databases can be added to the production system and queried against while only existing on a portion of the cluster.

6.2 Treating BLAST Databases Atomically

In BioCompute, the BLAST databases are treated as atomic; a complete copy of each database is stored locally on each node, and the split of the original input sequence is compared against the whole copy of the database. Strictly speaking, each BLAST database is not atomic. It it possible to split a database along sequence boundaries and compare input splits against database splits. Treating databases as atomic elements, however, greatly reduces the complexity of the "reduce" step for the BLAST abstraction: the original BLAST algorithms use domain-specific heuristics to determine the relevance of results from the comparison between an input sequence and the database, and order results accordingly. By maintaining the BLAST databases intact and whole, BioCompute avoids the necessity of reimplementing these heuristics. Each input sequence will have been compared against the whole database and the result set will already be ordered according to domain relevance. In addition, treating databases as atomic elements simplifies other processes including adding and distributing databases, distributing jobs among nodes, and adding nodes to the BioCompute pool.

7 Performance

7.1 Expectations

The BLAST algorithm compares each input sequences against all sequences in a database, and the comparisons of each input sequence are independent from the comparisons of every other sequence. Because of this decoupled nature of the sequence comparisons, it is trivial to distribute sequences to different computers to execute. It is expected that runtime increases linearly in proportion to the number of input sequences, not considering job startup time. Reducing the number of sequences in a job should therefore effect a roughly proportional linear reduction in runtime.

7.2 Preliminary Results

To test these expectations, an incoming query file was divided into roughly equal parts size for submission to Condor, a distributed platform. A 2.6GB version of the *nr* database was fully replicated onto 16 machines using the Chirp file-system so as to avoid multiple processes accessing the same database over the network. In order to measure performance increase, query files with 1,2,3..239,240 queries were submitted to *Condor Blast*. The total processing time of each query set was measured and recorded.

Referring to Figure 9:

- *Condor Blast* indicates runtimes for queries distributed as described
- *Blast* is running the n-query job against a single instance of the database residing elsewhere on the network
- *Parrot Blast* is running a single, undivided n-query job against a locally-stored copy of the database

This test shows that *Condor Blast* does indeed offer performance benefits over using normal BLAST; The runtime for *Condor Blast* remained relatively constant with the

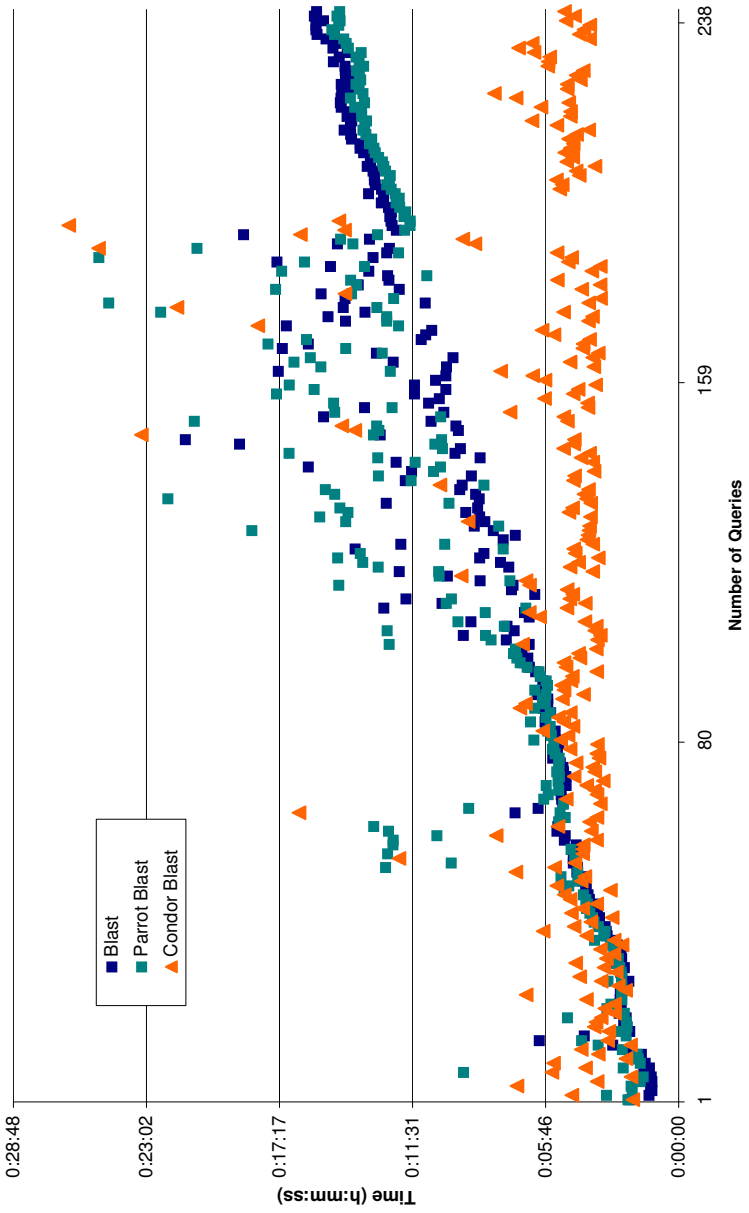

Figure 9: Preliminary Performance of BLAST distributed sequence-wise over Condor

increase in query set size, while both *Parrot Blast* and *Blast* increased linearly with query set size. These performance benefits are shown to scale while the number of Condor jobs submitted is smaller than the number of available nodes. The largest query set in this test was 240 queries, and incoming query files were split into batches of 30 queries per job. This results in only 8 machines being utilized at a time. On a 16-machine cluster, the trend for the *Condor Blast* line should in theory hold until 480 queries before stepping upwards because multiple jobs would have to be run on the same machine.

7.3 Characterizing Job Executions

Both the input division and results merging steps of a BioCompute job are currently implemented in linear time algorithms with respect to the number of queries submitted. With a small number of queries, these time taken by these steps are negligible, and the time has not been observed to be significant in any jobs submitted yet. If required in the future, these steps can be implemented as additional Condor jobs to be executed in a remote, distributed fashion.

7.4 Task Granularity

The number of sequences, N, in each division of sequences is a variable to tweak. Assuming low startup overhead time per job, a lower N is desirable. In an real-world environment, however, many factors could affect an optimal N, which could depend on the current job load on Condor and the amount of churn in Condor jobs—executing on a node with the needed sequence database already loaded in memory from a previous job might not need to read as much from disk and thus incur a lower startup penalty.

A 580-sequence input file was tested on BioCompute with values of 25, 50, and 100 for N, spawning 24, 12, and 6 individual Condor jobs, respectively. As shown in Table 2, the jobs with N = 25 ran much faster than the jobs with higher N values. The jobs with N = 50 and N = 100 had similar execution times.

N	CPU Time (s)	Wall Clock Time (s)	Jobs Spawned
10	6,523	266	58
25	5,230	327	24
50	5,000	723	12
100	4,368	790	6

Table 2: Execution time and job granularity for 580 input sequences.

N	CPU Time (s)	Wall Clock Time (s)	Jobs Spawned
10	29,855	1282	213
25	26,046	1058	86
50	24,291	1151	43
200	23,036	2891	11

Table 3: Execution time and job granularity for 2126 input sequences.

Runtime dramatically improves when N is sufficiently small to effectively utilize the entire cluster of BioCompute machines. These results indicate that a value for N is desired wherein enough jobs will be spawned to saturate the available nodes.

Additionally, a 2126-sequence input file was tested with values of 10, 25, and 200 for N. Table 3 showed a large speed improvement from N = 200 to N = 25, from 2891 seconds to 1058 seconds. However, when N = 10, 213 jobs were spawned and execution took 1282 seconds, indicating a general location of the lower limit for the optimal value of N.

Peripherally, choosing smaller N would lead to more granular status reporting: Condor maintains a logfile in the master directory indicating how many jobs have been spawned and their statuses. Lower values of N would enable more granular status reporting to be presented to the user using only Condor job statuses. In the event of unreliable systems, the additional granularity of smaller jobs would result in less wasted computing in the event of failure of an executing node.

7.5 Future Work

Many variables potentially relating to distributed BLAST performance have not been examined in this context. The size of the database remained constant among tests, but changing it may change the scalability characteristic of job execution [12]. In addition, the BLAST output formats differ widely in size and data layout, and these may have an effect on the total job execution time and the execution profile of the division and reduce/merge stages. In addition, these tests were conducted sequentially and in isolation from one another, but in a shared computing environment with other jobs. This would not be a typical usage pattern; users of BioCompute are expect to submit long jobs which would execute concurrently. Testing two jobs executing in tandem could show that multiple BioCompute jobs with small input query sets could effectively harness BioCompute by selecting different nodes on which to work. This could also potentially reveal inefficiencies where nodes alternate between processing two different BioCompute jobs on different databases, potentially causing a higher cache miss rate on memory. Also, BioCompute has processed jobs orders of magnitude larger than the test cases presented here, and tests should be conducted on jobs such as these to determine their execution profile.

8 Usability

8.1 Relevant Usability Paradigms

A primary design consideration of BioCompute is its usability and user-friendliness. During its development, two usability metaphors were used as guiding principles: electronic lab notebooks and portals. In this context, electronic lab notebooks seek to improve upon their paper-based brethren while portals seek to hide the perceived complexity of underlying systems. This section will discuss how BioCompute was implemented to achieve the desirable aspects of electronic lab notebooks and portals while recognizing and mitigating their drawbacks.

8.2 Portal Abstraction for Distributed Resources

8.2.1 Hiding Complexity

BioCompute aims to lower barriers surrounding distributed computing resources by presenting them for use in the form of a web portal. While such resources already exist, existence is not the sole prerequisite for effective use. By providing an easy, streamlined, and simplified means of access to pre-existing resources—in this case, the Condor pool at the University of Notre Dame—a web portal act as a catalyst for the production of meaningful results.

A portal for bioinformatics makes existing distributed computing resources accessible to researchers whose domain research can utilize such resources but whose domain expertise does not cover the effective engagement of such resources. BioCompute succeeds where it is an abstraction for the distributed resources it provides; if the user regards the system only ever as a "faster, easier BLAST that keeps track of things for me", then it may be considered to be working well.

In addition to hiding the complexity of distributed resources, a portal can mask the complexity of underlying domain applications. Many "wet sciences" researchers are often

not familiar or comfortable with command-line interfaces. Such interfaces, while efficient for those proficient in their use, often do not lend themselves to discoverability and do not meet the usability expectations of those whose background is not computer-based. In its original form BLAST is a command-line tool. BioCompute can make empower researchers to use BLAST by hiding the command-line behind a web interface, which is more familiar to typical modern computer users.

8.2.2 "Leaky" Abstractions

All abstractions are "leaky" to some degree [7]—that is, at times they break down and expose details which should be abstracted. Inasmuch as BioCompute is a layer of abstraction, it sits atop and depends upon a number of other independent, existing systems. There are a number of dangers which present themselves in such a situation. Users of BioCompute should not have to worry about the state of the distributed resources upon which it runs, nor should they even need to know it runs in a distributed environment. In the event of a failure in an underlying system, the abstraction "leaks" and users are affected.

8.2.3 User Interface Issues

The portal must accommodate users with varying levels of familiarity with the underlying system while minimizing user frustration on all levels. Often, novice users are discouraged by poor documentation or vague choices, while experienced users feel constrained by seemingly rigid choices and feel unproductive while learning a new environment. BioCompute should not be frustrating to users who are familiar with the BLAST command-line tool. While BioCompute should be inviting to users with less familiarity with computers than its developers, it should not rebuff users who require additional flexibility in order to effectively use the system. BioCompute delivers unique benefits in its harnessing of distributed resources and transcription features, and forcing advanced users to choose

between those benefits and additional flexibility in the domain application should be avoided.

8.2.4 Sequence Database Management

BioCompute will only be useful to researchers if it hosts the databases they are interested in querying.

In order for BioCompute to be a useful and practical resource for bioinformatics researchers, it must have relevant sequence databases against which to run. Inasmuch as it is BioCompute's goal to be an eminently useful and flexible tool, hosting databases which further this goal is desirable. So far, these databases have been sourced from public outlets, i.e. NCBI, or from the bioinformatics researchers themselves. Since there is currently a small group of users who are in close contact with those developing BioCompute, there has been little issue collaborating and sharing data to host. It would be desirable to streamline the sequence database submission process so that in the future user requests for hosting databases can be accommodated quickly. There is concern, however, with accepting dataset submissions and hosting and distributing automatically. To this end, a set of scripts have been developed to enable easy addition of databases to the BioCompute system, but which require explicit administrator action to do so.

Currently, the initial task of getting a database hosted on BioCompute presents additional burden to users compared to executing BLAST on their local machine. However, this should be a relatively infrequent activity for users; a typical use case would be a researcher conducting a large number of queries against a small number of databases over a period of time. In this case, the benefits of harnessing the distributed system are obvious and would outweigh the initial burden getting the database hosted. In addition, there may be future scenarios where in which it is necessary to limit the number of databases a user may upload. Regardless, development of BioCompute must occur with the realization of this matter.

8.2.5 Input File Limitations

A specific drawback of BioCompute in its current implementation rests in its use of HTTP file uploads for the sourcing of FASTA input sequences. There is currently an upper limit to the size of the input file which depends on the user's Internet connection and network proximity to the server and the HTTP server's timeout settings. Although comprehensive tests have not been conducted, files as large as 53 megabytes have been uploaded through a direct connection on the Notre Dame network. The server's timeout length has since been to facilitate larger input files.

8.2.6 BLAST Results File Formatting

The results returned from BioCompute differ slightly from those returned from BLAST. Currently, the results aggregation algorithm is a naïve ordered concatenation. This provides results which do not differ significantly from BLAST, but contain formatting differences which could be distracting and betray the abstraction of the distributed system. Developing a results post-processor to reconcile these differences is complicated by the fact that BLAST has multiple output formats which are neither isomorphic nor equivalent.

8.2.7 Mitigation

Many of these issues could be addressed by providing a command-line client to interface with the BioCompute system. By providing such a client, one could separate the interfaces used by advanced and regular users so as to minimize frustration in both groups and focus on the needs and expectations of each group separately. It could address the input sequence size limitation by providing a less restrictive file-upload delivery vector.

8.3 Electronic Notebook

BioCompute currently provides text-based "blank-page" [8] electronic lab notebook functionality for BLAST jobs; it allows users to add textual details about a job. This allows researchers to tightly couple their observations and notes about the job with the input, results, and recorded metadata of the job itself.

It is important that an electronic lab notebook be an unobtrusive element in research workflow; the use of a single component should not materially impinge on existing workflows nor set unreasonable restrictions on defining new processes. To this end, BioCompute's electronic lab notebook functionality has been of the "blank-page" philosophy; it provides room for free-form text input instead of limiting users to entering data in a rigid form. While many other data types are found and recorded in paper notebooks, including images, equations, graphs, and plots, BioCompute does not currently support them.

8.4 Automatic Transcription

Much bioinformatics research is conducted via computer-based experimentation, and as experiments become faster, larger, and easier to run, the feasibility of traditional paper-based lab notebooks reaches its scalability limit. An electronic lab notebook can especially aid in transcription of experiment details in a computer-based milieu.

Non-computer-based research traditionally required manually maintaining detailed notes of conducted experiments, but computer-based research, such as that facilitated by BioCompute, could easily benefit from an electronic lab notebook. The parameters and results of such research are digitally sourced and can be automatically recorded by an electronic notebook and presented for annotation or observation, obviating the need for rote manual transcription of details which can be automatically recorded and reducing the possibility of human error or forgetfulness in transcription.

In short, electronic notebooks can reduce the amount of drudgery and accidental

complexity inherent in the processes of conducting computer-based research and allow researchers to focus instead on the problem domain.

8.5 Data Explosion

Even in the face of exponentially-growing datasets, the absolute financial costs surrounding computer-based bioinformatics experiments are decreasing. Additionally, BioCompute aims to ameliorate temporal limitations on bioinformatics through distributed processing. This decrease in barriers begets an increase in the frequency and iteration cycle of experiments. In an environment with greater processing resources, experiments need less justification for execution, and the consideration given each experiment before running it is decreased. The ability and opportunity to conduct relatively rapid iterations of experiments at lower costs highlights the benefits of managing the resultant data and automatically transcribing the details and results of each job. In addition, the exponential growth of datasets provides a greater domain over which experiments can be run, adding further to the volume of bioinformatics data to be managed.

8.6 Collaboration and Sharing Results

In environments where research has scaled beyond one person, effective collaboration is limited by the physical singularity and location of a paper-based lab notebook. An electronic lab notebook can aid in the effective dissemination of experiment data throughout a research team. BioCompute provides functionality which lets researchers easily share their queries and associated notes with other BioCompute users.

8.7 Reproducibility

BioCompute is expected to regularly undergo changes, revisions, and updates to its execution model. It is important to enable the reproducibility of jobs executed within

BioCompute in the face of these changes. In order to ensure against these likely future changes, and additionally to ensure system modifications in BioCompute's testing environments do not affect adversely affect data processing in production environments, a snapshot of the scripts used to execute the job is stored alongside the results. This snapshot provides a trace of how the job was executed and records the command-line call to the BLAST executable and the surrounding computing environment.

In addition to being affected by BioCompute's script structure, reproducibility and an accurate log of research queries is affected by the possibility of changing databases on BioCompute's system. It is anticipated that as BioCompute grows in use, researchers will request custom sequence databases to be included in the system. In addition, some sequence databases currently hosted on BioCompute are publicly sourced and regularly updated, e.g. *nr*, the standard non-redundant database maintained by the NCBI.

In order to maintain strict reproducibility of results, ideally sequence databases should be static and not updated or removed once included into the BioCompute system. If an update to a database were to be included, it would be added under a related name with a differentiating timestamp, while still including the full copy of the earlier version.

This model is not universally feasible in the long-term, especially considering the continued growth of datasets. In addition, keeping all old datasets may clutter the user interface and make it difficult to find and select the desired database. However, if the database against which a job was run was not available or modified since the job was run, then the integrity of the lab notebook and the ability to confirm results is affected.

There are special cases of sequence databases which should be update-able in place without negatively affecting reproducibility. These databases should be publicly transparent in the criteria by which sequences are added and should be modified in an append-only mode. *nr* fits these criteria, as do similar databases from the NCBI, while custom datasets sourced from individual researchers or labs do not have these attributes. As such, a scenario is foreseeable wherein datasets with these attributes are updated in

place frequently, while other sequence datasets remain static and are updated in full under a differentiating timestamp.

8.8 Future Work

By providing these functionalities, BioCompute provides a platform for structured storage of data in a consistent interface, as opposed to a tool with an ephemeral, unstructured, and inconsistent relationship with the data upon which it operates. BioCompute automatically records meta-data associated with the job and stores it alongside the results of the BLAST program, thereby giving context and meaning to the results and facilitating reproduction and verification of the data. In the future, BioCompute can be expanded to provide these benefits to other types of data and for programs other than BLAST.

There are many possibilities for BioCompute to expand and become more useful in its function as an electronic lab notebook. First, adding fine-grained access controls similar to Unix groups would allow collaborating researchers to share results and progress without showing giving the results visibility to the entirety of BioCompute. While this may be antithetical to a wholly collaborative environment, it would improve usability by alleviating researchers' worries about the exposure of proprietary and custom datasets and results. Similar constructs could be added around user-sourced sequence databases.

BioCompute could be expanded to accommodate other types of data and other programs. InterProScan is another program which would fit BioCompute's model well. In addition, BioCompute could provide a flexible tool for chaining processes together in a manner conceptually similar to the stream-pipe functionality found in Unix command lines. A first step in this direction would be the ability to return a FASTA file of the sequences matched from a BLAST query (a normal BLAST query returns just the names of the matched sequences).

BioCompute could also serve as a data-processing workbench. A model could be constructed wherein data from experiments is sent first to BioCompute for hosting and

further processing options. This would provide a common and familiar environment where users could use common metaphors to manipulate, process, and add value to disparate data types.

9 Conclusion

Bioinformatics datasets are growing quickly and are large enough to effectively utilize distributed computing resources. Distributed computing presents many unique challenges and is complicated enough to benefit from layers of abstraction between the problem domain and the underlying system. Efforts should be made to make systems usable and accessible to users with typical computer skills.

The implementation of distributing a task among computers has a significant impact on performance, and the runtime improvements from distribution of a task are usually separate and orthogonal from the domain algorithm performance improvements.

BioCompute has laid the groundwork for a system which can effectively present distributed resources to end users. Development will continue on adding new application modules to the system to enable distributed execution of programs other than BLAST. Through the development of the service, areas for improvement were uncovered, including the need for a better file input management system, the advantages of providing a command-line client, and the concept of building a noun-verb framework in which data set are represented by nouns which are produced and consumed by distributed applications.

10 Bibliography

References

[1] E. Afgan and P. Bangalore. Performance characterization of blast for the grid. In *Proceedings of the 7th IEEE International Conference on Bioinformatics and Bioengineering*, pages 1394–398, 2007.

[2] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215, 1990.

[3] R. Braun, K. Pedretti, T. Casavant, T. Scheetz, C. Birkett, and C. Roberts. Three complementary approaches to parallelization of local blast service on workstation clusters. 1662, 1999.

[4] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large cluster. In *Operating Systems Design and Implementation*, 2004.

[5] H. Lin, X. Ma, P. Chandramohan, A. Geist, and N. Samatova. Efficient data access for parallel blast. In *IEEE International Parallel and Distributed Processing Symposium*, 2005.

[6] NCBI. Growth of genbank. `http://www.ncbi.nlm.nih.gov/Genbank/genbankstats.html`, February 2009.

[7] J. Spolsky. The law of leaky abstractions. April 2002.

[8] K. T. Taylor. The status of electronic laboratory notbooks for chemistry and biology. *Current Opinion in Drug Discovery and Development*, 9:348–353, 2006.

[9] D. Thain and M. Livny. Parrot: Transparent user-level middleware for data-intensive computing. In *Proceedings of the Workshop on Adaptive Grid Middleware*, New Orleans, September 2003.

[10] D. Thain, C. Moretti, and J. Hemmes. Chirp: A practical global filesystem for cluster and grid computing. Technical report, Journal of Grid Computing, 2009.

[11] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: the condor experience. *Concurrency: Practice and Experience*, 17:323–356, 2005.

[12] H. Yuasa, T. Iwasaki, M. Kurokawa, T. Shigetani, T. Horiki, and R. Himeno. Development of user-friendly supercomputing portal in bio research field. In *First International Conference on e-Science and Grid Computing*, 2005.