2019

Domain-specific language and infrastructure for genomics

Hamid Bagheri
Iowa State University

Domain-specific language and infrastructure for genomics

by

Hamid Bagheri

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Science

Program of Study Committee:
Hridesh Rajan, Major Professor
James Reecy
Samik Basu

The student author, whose presentation of the scholarship herein was approved by the program of study committee, is solely responsible for the content of this thesis. The Graduate College will ensure this thesis is globally accessible and will not permit alterations after a degree is conferred.

Iowa State University

Ames, Iowa

2019

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENTS

I would like to thank my friends and family for their support during my graduate research. I further thank Dr. Hridesh Rajan, my major advisor, for his guidance, patience, and support throughout this project. I would like to thank my committee members Dr. Samik Basu and Dr. James Reecy for their efforts and contributions to this thesis. I would also like to thank Dr. Andrew Severin for all the training in biology and computational insight toward this thesis. I also want to thank other Genome Information Facility members at Iowa State University for their feedback and comments. I want to thank Dr. Usha Muppirala for her initial work on the Boa$_g$ project and Dr. Rick Masonbrink for his feedback.

The content of this thesis is based on a published BMC Bioinformatics paper, 2019 [6]. This thesis was supported in part by grants from the National Science Foundation (NSF) under Grant CCF-15-18897, CNS-15-13263, and the VPR office at Iowa State University.

ABSTRACT

Creating a scalable computational infrastructure to analyze the wealth of information contained in data repositories is difficult due to significant barriers in organizing, extracting and analyzing relevant data. Shared data science infrastructures are needed to efficiently process and parse data contained in large data repositories [30]. This thesis introduces Boa$_g$, Boa for genomics. The main features of Boa$_g$ are inspired from existing languages for data-intensive computing and can easily integrate data from biological data repositories.

As a proof of concept, Boa$_g$, has been implemented to analyze RefSeq's 153,848 annotation (GFF) and assembly (FASTA) file metadata. Boa$_g$ provides a massive improvement from existing solutions like Python and MongoDB, by utilizing a domain-specific language that uses Hadoop infrastructure for a smaller storage footprint that scales well and requires fewer lines of code. We execute scripts through Boa$_g$ to answer questions about the genomes in RefSeq. We identify the largest and smallest genomes deposited, explore exon frequencies for assemblies after 2016, identify the most commonly used bacterial genome assembly program, and address how animal genome assemblies have improved since 2016. Boa$_g$ databases provide a significant reduction in required storage of the raw data and a significant speedup in its ability to query large datasets due to automated parallelization and distribution of Hadoop infrastructure during computations.

In order to keep pace with our ability to produce biological data, innovative methods are required. The Shared Data Science Infrastructure, Boa$_g$, provides researchers greater access to researchers to efficiently explore data in new ways. We demonstrate the potential of the domain-specific language Boa$_g$ using the RefSeq database to explore how deposited genome assemblies and annotations are changing over time. This is a small example of how Boa$_g$ could be used with large biological datasets.

CHAPTER 1. INTRODUCTION

In this chapter, we present the background of computational infrastructure, potential benefits to biology, and also introduce Boa$_g$ language and infrastructure.

1.1 Background

As sequencing data continues to accumulate in the online repositories [32], scientists can increasingly use multi-tiered data to better answer biological questions. A major barrier to these analyses lies with attaining a scalable computational infrastructure that is available to domain experts with minimal programming knowledge. The lengthy-time investment required for data wrangling tasks like organization, extraction, and analysis is increasing and is a well-known problem in Bioinformatics [36]. As this trend continues, a more robust system for reading, writing and storing files and metadata will be needed.

This can be achieved by borrowing methods and approaches from computer science. Boa$_g$ is a language and infrastructure that abstracts away details of parallelization and storage management by providing a domain-specific language and simple syntax [26]. The main features of Boa$_g$ are inspired by existing languages for data-intensive computing. These features include robust input/output, querying of data using types/attributes and efficient processing of data using functions and aggregators. Boa$_g$ can be implemented inside a Docker container or as a Shared Data Science Infrastructure (SDSI) [30]. Running on a Hadoop cluster [17], it manages the distributed parallelization and collection of data and analyses. Boa$_g$ can process and query terabytes of raw data. It also has been shown to substantially reduce programming efforts, thus lowering the barrier of entry to analyze very large data sets and drastically improve scalability and reproducibility [17]. Raw data files are described to Boa$_g$ with attribute types so that all the information contained in the raw data file can be parsed and stored in a binary database. Once complete, the reading, writing, storing and querying the data from these files is straightforward and efficient as it creates a dataset

that is uniform regardless of the input file standard (GFF, GFF3, etc). The size of the data in binary format is also smaller.

1.2 Potential for data parallelization framework in biology

There are several very large data repositories in biology that could take advantage of a biology specific implementation of Boa$_g$: The National Center for Biotechnology Information (NCBI), The Cancer Genome Atlas (TCGA), and the Encyclopedia of DNA Elements (ENCODE). NCBI hosts 45 literature/molecular biology databases and is the most popular resource for obtaining raw data for analysis. NCBI and other web resources like Ensembl are data warehouses for storing and querying raw data, sequences, and genes. TCGA contains data that characterizes changes in 33 types of cancer. This repository contains 2.5 petabytes of data and metadata with matched tumor and normal tissues from more than 11,000 patients. The repository is comprised of eight different data types: Whole exome sequence, mRNA sequence, microRNA sequence, DNA copy number profile, DNA methylation profile, whole genome sequencing, and reverse-phase protein array expression profile data. ENCODE is a repository with a goal to identify all the functional elements contained in human, mouse, fly, and worm. This repository contains more than 600 terabytes (personal communication with @EncodeDCC and @mike_schatz) of data with more than 40 different data types with the most abundant data types being ChIP-Seq, DNase-Seq, and RNA-Seq. These databases represent only the tip of the iceberg of potentially large data repositories that could benefit from the Boa$_g$ framework. While it is common to download and analyze small subsets of data (tens of Terabytes for example) from these repositories, analyses on the larger subsets or the entire repository are currently computationally and logistically prohibitive for all but the most well funded and staffed research groups. While BioMart [33], Galaxy, and other web-based infrastructures provide an easy to use tool for users without any knowledge in programming to download subsets of the data, the needs of the advanced users using the entire database aren't met as evidenced by a plethora of bash scripts, R scripts and Python scripts that are widely utilized and reinvented by bioinformaticians. Retrieving the genomics data and performing data-intensive computation can be challenging using existing APIs. Biomartr [13] is an R package to retrieve raw genomics data that try to minimize some of this complexity.

Here we discuss an initial implementation of Boa for genomics on a small test dataset, NCBI Refseq, a database containing data and metadata for 153,848 genome annotation files (GFF). We show the potential of Boa$_g$ in a comparative context with Python and MongoDB by assessing various statistics of the Refseq database and answer the following four questions.

- What is the smallest and largest genome in RefSeq?
- How has the average number of exons per gene in genomes of a clade changed for genomes deposited before and after 2016?
- How has the popularity of the top five assembly programs in bacteria changed over time?
- How has assembly quality changed for genomes deposited before and after 2016?

CHAPTER 2. REVIEW OF LITERATURE

Genomics-specific languages and libraries are common in high-throughput sequencing analysis such as S3QL, which aims to provide biological discovery by harnessing Linked Data [12]. In addition, there are libraries like BioJava [28], Bioperl [34], and Biopython [9] that provide tools to process biological data.

MongoDB is an open-source NoSQL database that also supports many features of traditional databases like sorting, grouping, aggregating, indexing, etc. MongoDB has been used to handle large scale semi-structured or NoSQL data. Datasets are stored in a flexible JSON format and therefore can support data schema that evolves over time. MapReduce [10] is a framework that has been used for scalable analysis in scientific data. Hadoop is an open-source implementation of MapReduce. In the MapReduce programming model, mappers and reducers are considered as the data processing primitives and are specified via user-defined functions. A mapper function takes the key-value pairs of input data and provides the key-value pairs as an output or input for the reduce stage, and a reducer function takes these key-values pairs and aggregates data based on the keys and provide the final output. Some organizations have used the power of MongoDB and Hadoop framework together [3] to address challenges in Big Data. Genomics England [2] runs the 100,000 Genomes Project [38] using MongoDB to harness huge amount of data in Bioinformatics. There are also several tools in the field of high-throughput sequencing analysis that use the power of the Hadoop and MapReduce programming model. Heavy computation applications like BLAST, GSEA and GRAMMAR have been implemented in Hadoop [35]. SARVAVID [25] has implemented five well-known applications for running on Hadoop: BLAST, MUMmer, E-MEM, SPAdes, and SGA. BLAST [5] was also rewritten for Hadoop by Leo *et.al.* [23]. In addition to these programs, there are other efforts based on Hadoop to address RNA-Seq and sequence alignment [27, 31, 22].

A significant barrier to utilize the Hadoop framework in Bioinformatics is the difficulty of the interface and the amount of expertise that is needed to write MapReduce programs [4]. The proposed work tries to abstract away the details of these complexities and open a door for more Bioinformatics applications.

Most applications could be called from MapReduce rather than reimplementing them. Unfortunately, there currently does not exist a tool that combines the ability to query databases, with the advantage of a domain-specific language and the scalability of Hadoop into a Shared Data Science Infrastructure for large biology datasets.

Boa, on the other hand, is such a tool but is currently implemented for mining very large software repositories like GitHub and Sourceforge [16, 15, 7]. Boa features are inspired by object-oriented visitors and provide a default depth-first traversal strategy [19]. It has been utilized to mine billions of Abstract Syntax Tree (AST) nodes to study usages of Java language features over time [18].

Upadhyaya *et al.* proposed a new direction for accelerating ultra-large-scale mining based on task-specific similarity [39] and utilized Boa language and its infrastructure. Later, they proposed collective program analysis (CPA), a technique for the large-scale source code analyses by leveraging analysis specific similarity [40]. This work proposed a technique that reduces the amount of computation performed by the ultra-large-scale source code mining task. They showed a significant improvement over the current works. In another work, Upadhyaya *et al.* used Boa language and its dataset to accelerate source code analysis such as control flow analysis, data flow analysis at large scale [41].

Tiwari *et al.* proposed Candoia, a platform for building and sharing Mining Software Repositories(MSR) tools that is an extension of Boa language [37]. There have also been several large-scale empirical studies by using Boa, for example, Maddox *et al.* studied Substitutability in the Presence of Effects [24] and Zhang *et al.* leveraged Boa to traverse the abstract syntax trees (ASTs) of Java files and studied the API misuse [42].

Dyer *et al.* showed task fusion on optimizing Boa infrastructure by which automatically merges multiple tasks into a single task. He showed that this optimization provides up to 10 times faster execution time compared to running the tasks individually [14].

Boa also recently has been applied to address the potentials and challenges of Big Data in transportation [20] and for genomics [6] that is described in this thesis.

CHAPTER 3. METHODOLOGY

This chapter presents our methods of dataset choice, dataset generation, and design and implementation choices of Boa$_g$. This chapter also shows the overview architecture of the proposed works. We will also describe how Boa$_g$ could be integrated with current general-purpose languages like Python in Jupyter notebooks.

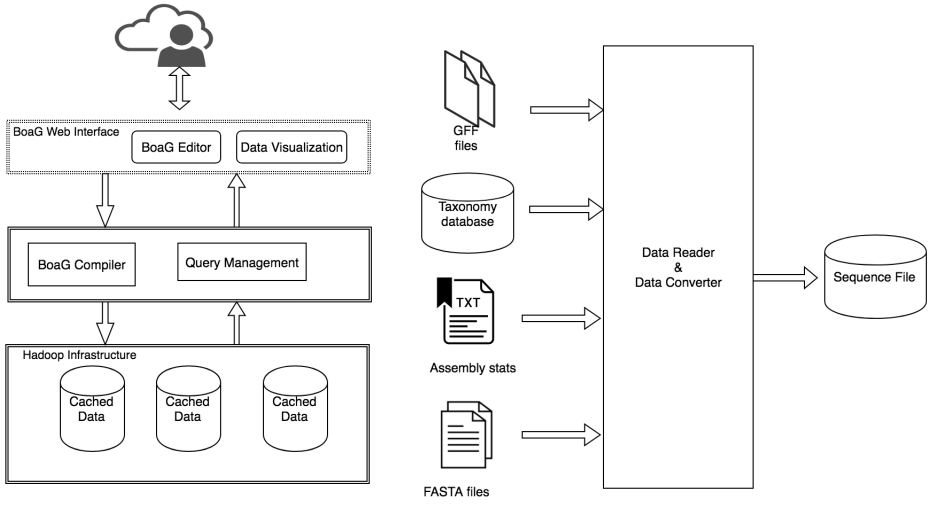
3.1 Choice of Biological repository for prototype implementation

RefSeq is a relatively small dataset containing information on well-annotated sequences spanning the tree of life: plants, animals, fungi, archaea, and bacteria. The smaller database size permits rapid iterations of Boa$_g$ applied to biology and illustrates the benefits of a genomics specific language. RefSeq also has a decent amount of metadata about genome assemblies and their annotations for which as far as we know has not been explored as a whole. Unfortunately, due to the rapid advancement of sequencing technologies and genome assembly/annotation programs, deriving biologically meaningful information from comparisons of assembly stats across the entire dataset is not possible; however, as a demonstration of the usefulness of a Boa$_g$ infrastructure, we show how straightforward it is to ask questions about how the database and the metadata have changed over time which gives insight into how improvements in sequencing technology and assembly/annotation programs have affected the data contained in this repository. These types of information would be challenging to procure directly from the online repository.

3.2 Design and implementation considerations

As a domain-specific language, careful consideration must be taken in its design for Hadoop based infrastructure implementation for RefSeq data. The overall workflow for Boa$_g$ requires a program written in Boa$_g$ that is submitted to the Boa$_g$ infrastructure (Figure 3.1 (a)). The infrastructure takes the submitted program and compiles with the Boa$_g$ compiler and executes the program on a distributed Hadoop cluster

using a Boa$_g$ formatted database of the raw data. Boa$_g$ has aggregators, which are functions that run on the entire or a large subset of the database to take advantage of the Boa$_g$'s database, which is designed to distribute both data and compute across a Hadoop cluster.



(a) An overview of Boa$_g$ Architecture (b) An overview of Data Generation in Boa$_g$

Figure 3.1 Boa$_g$ Architecture and Data Generation

A Boa$_g$ infrastructure provides the following benefits for exploring large datasets

- A computational framework on top of Hadoop that can query large datasets in minutes.
- An efficient data schema that provides storage efficiency and parallelization.
- An expandable database integration.
- A domain-specific language that can be incorporated in a container, Galaxy framework or along with any language like R or Python in a Juypter notebook.

3.2.1 Genomics-specific Language and data schema

To create the domain-specific language for biology in Boa$_g$, we created domain types, attributes and functions for the RefSeq dataset that includes the following raw file types: FASTA, GFF and associated metadata, as shown in Table 3.1, Genome, Sequence, Feature, and Assembler are types in Boa$_g$ language

Table 3.1 Domain types for Genomics data in Boa$_g$

Type	Attributes	Details
Genome	taxid	Taxonomy ID of each species
	refseq	Refseq ID of the GFF file
	Sequence	List of sequence reads in each GFF file[1].
	AssemblerRoot	List of assembly programs associated with this genome
Sequence	accession	Accession number
	header	Header of Sequence
	FeatureRoot	List of features including exon,gene,mRNA, and CDS associated with this sequence
	seq	Actual DNA sequences from FASTA files
FeatureRoot	refseq	This field shows the key ID
	feature	This field is the list of features associated with this ID
Feature	accession	Accession code of the Sequence
	seqid	Sequence ID
	source	A text qualifier that describes the algorithm or procedure that generated this feature.
	ftype	Type of the feature
	start	starting point of the feature
	end	End point of the feature
	score	Score of the feature. This is a floating point number.
	strand	+ and - for positive and negative strand respectively
	phase	Phase of the feature. The phase is one of the integers 0, 1, or 2
	Attribute	List of attributes for each feature
	parent	Shows the parent of the attribute
Attribute	id	Attribute ID
	tag	Attribute tag including gbkey etc.
	value	Value of the tag
AssemblerRoot	Assembler	List of assembly programs
	total-length	Total length or genome size(base pair)
	total-gap-length	Total gap length after genome assembly
	scaffold-N50	Scaffold N50 metric
	scaffold-count	Scaffold count metric
	contig-N50	Contig N50 metric
	contig-count	Contig count metric
Assembler	name	Assembly program used to assemble the genome
	desc	Program attributes: program name, program version, etc.

Table 3.2 The Boa$_g$ aggregators list

Aggregator	Description
MeanAggreagtor	Calculates the average
MaxAggreagtor	Finds the maximum value
SumAggregator	Calculates the sum of the emitted values to the reducer
MinAggregator	Finds the minimum value
TopAggregator	Takes an integer argument and returns the top elements for the given argument
StDevAggregator	Calculates the standard deviation

and taxid, RefSeq, etc are attributes of the genome type. We created the data schema based on the Google protocol buffer, which is an efficient data representation of genomic data that provides both storage efficiency and efficient computation for Boa$_g$.

3.2.2 Output Aggregators in Boa$_g$

Table 3.2 shows the predefined aggregators in the Boa$_g$ language, for example, top, mean, maximum, minimum, etc. These aggregators are also available in traditional RDBS and MongoDB [8]; however, Boa$_g$ is flexible enough to define new aggregators. Boa$_g$ provides a specific type called output types that collect and aggregate data and provide a single result. When a Boa$_g$ script is running in parallel, it emits values to the output aggregator that collects all data and provides the final output. Aggregators can also contain indices that would be a grouping operation similar to traditional query languages.

3.3 Boa$_g$ database and new data type integration

The Boa$_g$ infrastructure is designed to fully utilize data parallelization facilities in Hadoop infrastructure. The raw data for file types and metadata was parsed into a Boa$_g$ database on top of a Hadoop sequence file (Figure 3.1 (b)). A compiler, file reader, and converter were written in Java to generate this database and are provided in the GitHub repository (https://github.com/boalang/bio/tree/master/compiler). To integrate a new dataset the data schema in protocol buffer format needs to be modified and a data reader in Java that reads the raw data, for example, GFF, TXT, Fastq, etc, is needed that can convert it to a binary format of Boa$_g$ database. An additional example is provided in the GitHub repository.

Boa$_g$ efficiency was tested on a shared Hadoop cluster on Bridges (https://portal.xsede.org/psc-bridges) with 5 nodes and up to 256 map tasks.

3.4 Data availability

All scripts, step by step process of scientific discovery, and additional examples of Boa queries used in this thesis can be found in our repository. The raw data files, Boa$_g$ database, and JSON MongoDB files can be obtained from an online repository (https://boalang.github.io/bio/). A Docker container with Boa$_g$ scripts, a Boa$_g$ sequence file of a subset of the raw files and instructions on how to use Boa$_g$ can also be downloaded from this location. We have generated a subset of GFF files and assembly statistics files for

all fungi data contained in RefSeq. This data subset is 5.4 GB and can be used to test Boa$_g$ queries for reproducible results.

3.5 Run Boa$_g$ on Docker container and Jupyter

For the fungal data subset, users can run a containerized version of a 3 node Hadoop cluster for Boa$_g$ as well as Jupyter versions on a single machine. These integrations with current technologies can help users test and run queries and reproduce our results. Instructions on how to run a Docker version and a Jupyter version of Boa$_g$ are available on this website: https://boalang.github.io/bio/.

3.6 Application of Boa$_g$ to the RefSeq database

A total of 153,848 annotations (GFF), assembly (FASTA) files, and metadata were downloaded from NCBI RefSeq [29] and written to a Boa$_g$ database. Metadata included genome assembly statistics (Genome size, scaffold count, scaffold N50, contig count, contig N50) and assembler software used to generate the assembly from which the genome annotation file was created.

CHAPTER 4. RESULTS

This chapter presents some examples of questions that could be asked on the RefSeq dataset. It also worth mentioning that these are a few examples, the infrastructure provides an interface that users can test different hypotheses.

4.1 Summary statistics of RefSeq

While it is straightforward to use the RefSeq website (https://www.ncbi.nlm.nih.gov/refseq/) to look up this information for your favorite species, it is cumbersome to look up this information for tens to hundreds of species. Similarly, while each of these genomes have an annotation file, querying and summarizing the information contained in this annotation file from several related genomes such as the average number of genes, the average number of exons per gene and average gene size requires downloading and organizing the annotation files of interest prior to calculating the statistics.

Data from the RefSeq database was downloaded, a schema was designed and a Hadoop sequence file generated for use with Boa_g, a domain-specific language and shared data infrastructure. The RefSeq data used in this first implementation of Boa$_g$ contains GFF files and metadata from bacterial (143,907), archaea (814), animal (480), fungal (284) and plant (110) genomes. Each genome has metadata related to the quality of its assembly (Genome size, scaffold count, scaffold N50, contig count, contig N50), the assembler software, and the genic data contained within the GFF annotation file.

Our goal was to implement Boa$_g$ on a biological dataset to demonstrate a means to explore large datasets. In the following subsections, we will answer the four questions posed in the introduction and explore Boa$_g$ efficiency in storage, speed, and coding complexity.

4.2 The largest and smallest genome in the RefSeq database

Researchers might be interested in finding what is the largest and smallest genome in RefSeq? As of February 16th, 2019, the largest genome in the RefSeq database was Orycteropus afer (aardvark, GCF_000298275.1) at a length of 4,444,080,527 bp. The smallest genome is RYMV, a small circular viroid-like RNA hammerhead ribozymein sequenced from Rice and annotated as a Rice yellow mottle virus satellite (viruses). Its complete genome has a length of 220 bases and has a RefSeq id GCF_000839085.1.

With the full RefSeq dataset in a Hadoop sequence file, this statistics only required seven lines of Boa$_g$ code (Figure 4.1). In line one, variable g is defined as a Genome which is a top-level type in our language. MaxGenome and MinGenome are output aggregators that produce the maximum and minimum genome length respectively. Lines five and seven in the code emit the assembly total length to the reducer for all the genomes in the dataset, then the reducer will identify the largest and smallest genomes. It took Boa$_g$ approximately 30 seconds to finish this query when using a single node without Hadoop. It took the equivalent query using Python approximately one hour using a single core.

```
1  g: Genome=input;
2  MaxGenome: output maximum(1) of string weight int;
3  MinGenome: output minimum(1) of string weight int;
4  asm:= getAssembler(g.refseq);
5  MaxGenome << g.refseq weight asm.total_length;
6  if(asm.total_length>0)
7     MinGenome << g.refseq weight asm.total_length;
```

Figure 4.1 Code to find the smallest and largest genomes in RefSeq.

4.3 Study the changes of average number of exons per gene over time

Researchers might also be interested in finding how has the average number of exons per gene in a species clade changed for genomes deposited before and after 2016? Due to the rapid advancement of sequencing technologies and genome assembly/annotation programs, any meaningful biological changes in gene and exon frequency will be confounded with these advancements. We explored seven clades: five kingdoms and two phyla to explore how exon number, gene number, gene length and exons per gene have changed before and after 2016. These branches of the tree of life included Bacteria, Archaea, Fungi, As-

comycota (a fungal phylum), Viriplantae (plants), Eudicotyledons (a clade in flowering plants) and Metazoans (a clade of animals). In the last two years, the number of sequenced bacterial genomes has nearly quadrupled, while all other clades have seen at least a 50% increase in the RefSeq database (Table 4.1, Table 4.2). The number of genes, number of exons and exons per gene have increased for all clades database (Table 4.1, Table 4.2). Since prokaryotes do not have exons, Bacteria and Archaea were excluded from this query for exon number and exon per gene (NA). A higher number of exons per gene for the Eukaryotes suggests that gene models are improving and becoming less fragmented. This improvement could be due to improvements in gene annotation software or assembly contiguity.

```
1  g: Genome = input;
2  geneCounts: output sum[string][string] of int;
3  exonCounts: output sum[string][string] of int;
4  adata := getAssembler(g.refseq);
5  asYear :=yearof(adata.assembly_date);
6  if(asYear >= 2016)
7      foreach(i:int; def(g.sequence[i])){
8         fdata:=getFeature(g.refseq,g.sequence[i].accession);
9         foreach(j:int; def(fdata.feature[j])){
10                 if(match("gene",fdata.feature[j].ftype))
11                    geneCounts [g.refseq][g.taxid]<< 1;
12                 if(match("exon",fdata.feature[j].ftype))
13                    exonCounts [g.refseq][g.taxid]<< 1;
14        }
15     }
```

Figure 4.2 Number of exons, genes, and exons per gene after 2016. The output is shown in Table 4.1.

We find fewer genes in archaea than in bacteria, at 2.9k and 4.3k genes respectively. The highest gene numbers in eukaryotes are plants (43k), with animals and fungi being having fewer genes at 24.9k and 10k, respectively [21]; however, the mean gene length for these clades has not changed between time points, indicating that the increased exon content per gene is likely due to an improvement in annotation software.

This query required 15 lines of Boa$_g$ code (Figure 4.2) using a five node shared Hadoop cluster on Bridges with 64 mappers approximately 42 minutes to answer this question. It took the equivalent query using 45 lines of Python code approximately 20 hours using a single core.

Table 4.1 Exon Statistics for years >= 2016

Name	Total species	Exon number	Gene number	Gene Length	Exon per Gene
Bacteria	92287	N/A	$4.3k \pm 1.5k$	890 ± 64	N/A
Fungi	90	$32.3k \pm 1.8k$	$10k \pm 3.5k$	$1.6k \pm 171$	2.9 ± 1.3
Archaea	338	N/A	$2.9k \pm 0.9k$	851 ± 31	N/A
Viridiplantae	46	$385k \pm 155k$	$43k \pm 21k$	$4.1k \pm 1.3k$	9.2 ± 1.9
Metazoas	185	$462k \pm 280k$	$24.9k \pm 10.3k$	$23k \pm 11.8k$	17.7 ± 6.4
Ascomycota	70	$28.4k \pm 13.7k$	$10.4k \pm 3.1k$	$1.6k \pm 142$	2.5 ± 0.8
eudicotyledons(dicots)	37	$397k \pm 167k$	$45k \pm 22k$	$3.8k \pm 688$	9 ± 1.3

Table 4.2 Exon Statistics for years < 2016

Name	Total species	Exon number	Gene number	Gene Length	Exon per Gene
Bacteria	51537	N/A	$3.8k \pm 1.5k$	885 ± 65	N/A
Fungi	194	$29k \pm 20k$	$9.2k \pm 3.5k$	$1.6k \pm 254$	2.8 ± 1.5
Archaea	474	N/A	$2.9k \pm 0.8k$	855 ± 40	N/A
Viridiplantae	61	$273k \pm 153k$	$32k \pm 17k$	$4.1k \pm 2.3k$	8 ± 2.5
Metazoas	262	$314k \pm 211k$	$22.3k \pm 9.6k$	$22k \pm 12k$	13.4 ± 5.4
Ascomycota	143	$25.2k \pm 14.3k$	$9.5k \pm 3.1k$	$1.6k \pm 205$	2.4 ± 1
eudicotyledons(dicots)	41	$328k \pm 133k$	$38k \pm 16k$	$4k \pm 1.4k$	8.6 ± 1.3

4.4 Popularity of bacterial genome assembly programs

Another research question might be how has the popularity of bacterial genome assembly programs changed? The choice of genome assembly program to assemble a genome depends on many factors including but not limited to user familiarity of the program in the domain, ease of use, assembly quality, turnaround time. Looking at the number of genomes assembled by the top five most popular assemblers in bacteria indicate that more genomes are being assembled over time, that there was a brief period of popularity with AllPaths in 2014 and a rapid rise in popularity of the SPAdes assembler in the last couple of years. CLC workbench offers a GUI interface to users without programming experience and has consistently maintained a slice of the user market (Figure 4.4).

This query required six lines of Boa$_g$ code Figure 4.3 using a five node Hadoop cluster with 32 mappers approximately 30 seconds to answer this question. The equivalent single-cored Python query took approximately one hour with 35 lines of code.

```
1  g: Genome = input;
2  counts: output sum [int][string][string] of int;
3  asm := getAssembler(g.refseq);
4  asYear :=yearof(asm.assembly_date);
5  foreach(k:int; def(asm.assembler[k]))
6     counts [asYear][g.taxid][asm.assembler[k].name]<<1;
```

Figure 4.3 Bacterial assembly programs popularity over time. The output of this script is shown in Figure 4.4.

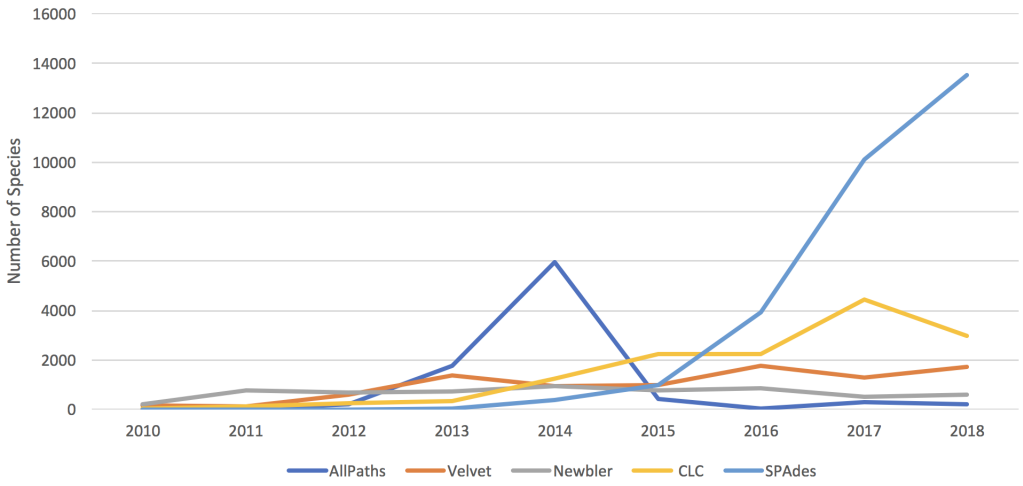


Figure 4.4 Assembler programs for Bacteria over the years

4.5 Study the quality of metazoan assembly programs over time

Another research question is how has metazoan assembly quality changed for genomes deposited before and after 2016? To minimize bias in organismal variation and assembly software, we have limited our comparison to metazoans and the top three assembly programs. The popular assembly programs for metazoans has been AllPaths after 2016 while SOAPdenovo was the most popular one before 2016. A high-quality assembly is characterized by a low scaffold count and high N50, stats that dramatically improved at the 2016 transition. As it can be seen in Table 4.3 and Table 4.4, the scaffold count has decreased for all three

assemblers after 2016 while the contig N50 metric has increased. This is not a surprise, as assembly algorithms are expected to improve over time. Newbler had a dramatic decrease in scaffold count after 2016. The highest average N50 among metazoans belongs to AllPaths.

This query, Figure 4.5, required 6 lines of Boa$_g$ code using five nodes Hadoop cluster with 32 mappers approximately 30 seconds. An equivalent single-cored Python query took approximately one hour and 32 lines of code

Table 4.3 List of top three most used assembly programs for Metazoa (Year > =2016)

Kingdom	Program Name	species	Total length	Scaffold-count	ScaffoldN50	ContigCount	ContigN50
Metazoa	SOAPdenovo	21	$1B \pm 0.8B$	$38k \pm 49k$	$7.8M \pm 11M$	$86k \pm 66k$	$98k \pm 208k$
	AllPaths	48	$0.9B \pm 0.7B$	$7.1k \pm 7k$	$4.3M \pm 1.4M$	$33k \pm 38k$	$188k \pm 335k$
	Newbler	7	$0.8B \pm 0.9B$	$3.3k \pm 2.2k$	$877k \pm 910k$	$56k \pm 80k$	$75k \pm 60k$

Table 4.4 List of top three most used assembly programs for Metazoa (Year < 2016)

Kingdom	Program Name	species	Total length	Scaffold-count	ScaffoldN50	ContigCount	ContigN50
Metazoa	SOAPdenovo	98	$1.2B \pm 0.7B$	$40k \pm 38k$	$4.5M \pm 13M$	$116k \pm 79k$	$42k \pm 48k$
	AllPaths	54	$1.5B \pm 1.1B$	$11k \pm 13k$	$7.4M \pm 9.7M$	$119k \pm 97k$	$38k \pm 32k$
	Newbler	18	$0.9B \pm 0.9B$	$87k \pm 117k$	$2.1M \pm 2.3M$	$133k \pm 157k$	$34k \pm 27k$

```
1  g: Genome = input;
2  counts: output collection [string][string][int][int][int][int][int][int] of int;
3  adata := getAssembler(g.refseq);
4  asYear :=yearof(adata.assembly_date);
5  if(asYear >= 2016)
6      counts[g.refseq][g.taxid][adata.total_length][adata.total_gap_length][adata.
   scaffold_count][adata.scaffold_N50][adata.contig_count][adata.contig_N50] <<1;
```

Figure 4.5 Assembly statistics for genomes for years after 2016. The output is shown in Table 5.1.

CHAPTER 5. DISCUSSION AND FUTURE WORK

In this chapter, we describe the storage and computational efficiency of Boa$_g$ infrastructure. We discuss and compare Boa$_g$ with Python and MongoDB. Finally, we summarize our work and provide suggestions for future work.

5.1 Database storage efficiency and computational efficiency with Hadoop

One benefit of the Boa$_g$ database is the significant reduction in the required storage of the raw data. The downloaded NCBI RefSeq data was 379GB but reduced to 64GB (6.2 fold reduction) in the Boa$_g$ database. This data size reduction is due to the binary format of the Hadoop Sequence file which makes disk writing faster than a text file (Figure 5.2). A fungi-only subset of the RefSeq data was dramatically reduced from 5.4GB to 0.5 GB (10 fold reduction). This variability in size reduction is presumably due to variability in the number and size of files among phyla.

The second benefit of Boa$_g$ is its ability to take advantage of parallelization and distribution during computation. Increasing the number of Hadoop mappers for a Boa$_g$ job decreases the query turnaround time. Taking the four queries we posed in the introduction, we varied the level of Hadoop mappers to show the speedup that results by adding additional Hadoop mappers to an analysis. Figure 5.1, demonstrates the exponential decrease in required computation time with a corresponding increase in the number of Hadoop mappers. As you can see, if the number of mappers is not optimized for the amount of computational infrastructure than the second query takes approximately 350 minutes on 2 mappers to complete; however, as more mappers are added, the time required levels out to less than one minute on assembly related queries. This lower bounds of this relationship is presumably due to the overhead of splitting and gathering of data across the mappers. As we add more mappers the running time decreases for example with 256 mappers runtime is 22 minutes on the entire RefSeq. It is not difficult to see the benefit of using a domain-specific language like Boa$_g$ and Hadoop infrastructure to query much larger biological datasets than RefSeq.

Table 5.1 Kingdoms and average summary statistics for their genome assemblies (Years > =2016)

Tax ID	Name	Species	Total length	Scaffold-count	ScaffoldN50	ContigCount	ContigN50
2	Bacteria	92290	$4.3M \pm 1.6M$	66 ± 78	$0.9M \pm 1.4M$	132 ± 176	$0.39M \pm 0.86M$
4751	Fungi	90	$29M \pm 15M$	139 ± 159	$1.3M \pm 0.9M$	360 ± 688	$0.78M \pm 1M$
2157	Archaea	338	$2.9M \pm 0.98M$	52 ± 40	$0.38M \pm 0.43M$	74 ± 121	$0.53M \pm 71M$
33090	Viridiplantae	46	$0.97B \pm 0.88B$	$9.1k \pm 18.3k$	$31M \pm 49M$	$38k \pm 43k$	$1.8M \pm 4.9M$
33208	Metazoas	185	$1.2B \pm 0.95B$	$20.6k \pm 43.7k$	$22M \pm 36M$	$53k \pm 77k$	$2.5M \pm 7.9M$
71240	eudicotyledons(dicots)	37	$0.91B \pm 0.76B$	$6.4k \pm 10.6k$	$26M \pm 50M$	$40k \pm 44k$	$1.6M \pm 4.3M$

Table 5.2 Kingdoms and average summary statistics for their genome assemblies (Years <= 2015)

Tax ID	Name	Species	Total length	Scaffold Count	ScaffoldN50	ContigCount	ContigN50
2	Bacteria	51962	$3.8M \pm 1.6M$	45 ± 82	$1.3M \pm 1.5M$	126 ± 177	$0.27M \pm 0.55M$
4751	Fungi	202	$29M \pm 17M$	341 ± 699	$2M \pm 1.7M$	858 ± 1433	$0.55M \pm 0.75M$
2157	Archaea	470	$2.9M \pm 1M$	17 ± 16	$1.35M \pm 1.17M$	110 ± 126	$0.38M \pm 0.7M$
33090	Viridiplantae	67	$0.62B \pm 0.68B$	$22.9k \pm 46.6k$	$14.7M \pm 24.9M$	$52.5k \pm 71.6k$	$0.47M \pm 1.8M$
33208	Metazoas	295	$1.3B \pm 1B$	$37.4k \pm 64.2k$	$7.2M \pm 14M$	$118.6k \pm 119k$	$0.13M \pm 1.2M$
71240	eudicotyledons(dicots)	46	$0.754B \pm 0.750B$	$26.3k \pm 53.5k$	$17M \pm 27M$	$58.8k \pm 74k$	$0.3M \pm 1.6M$

Taking advantages of Hadoop based infrastructure, all the queries in the Table 5.1 and Table 5.2 that describe the genome assembly statistics before and after 2016 transition required less than a minute.

5.2 Comparison between MongoDB and Boa$_g$

The data schema in MongoDB also needs to be saved along with the data, the output files are larger and take longer to write (Figure 5.2). The JSON file size is larger and on average it is more than double the size of the RefSeq raw data. While experts in MongoDB may write this query more efficiently, the Boa$_g$ language requires fewer lines of code (Figure 5.3), thereby providing an easier interface for bioinformaticians to explore big data.

The performance of MongoDB and Hadoop has been previously compared [11], showing that the read-write overhead of Hadoop has a lower read-write overhead (Table 5.3).

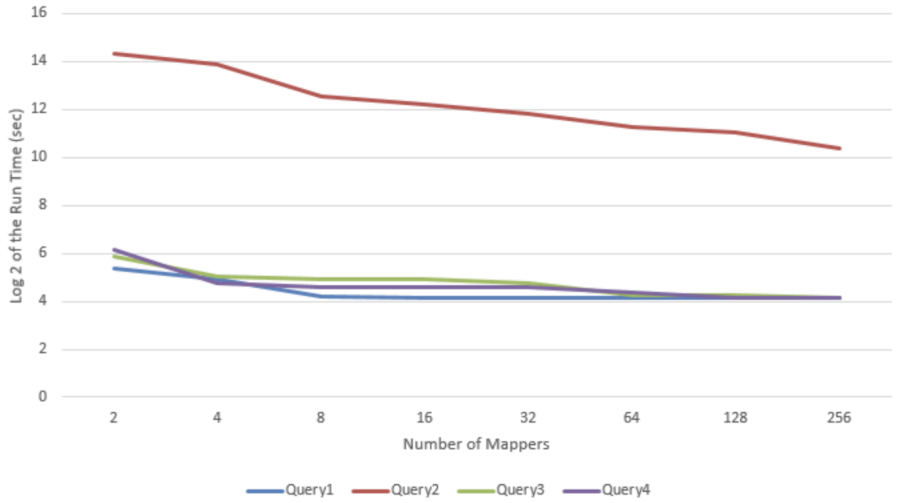


Figure 5.1 Scalability of Boa$_g$ programs (time is in Log base 2 (sec)). Queries 1,2,3 and 4 are the four questions investigated here.

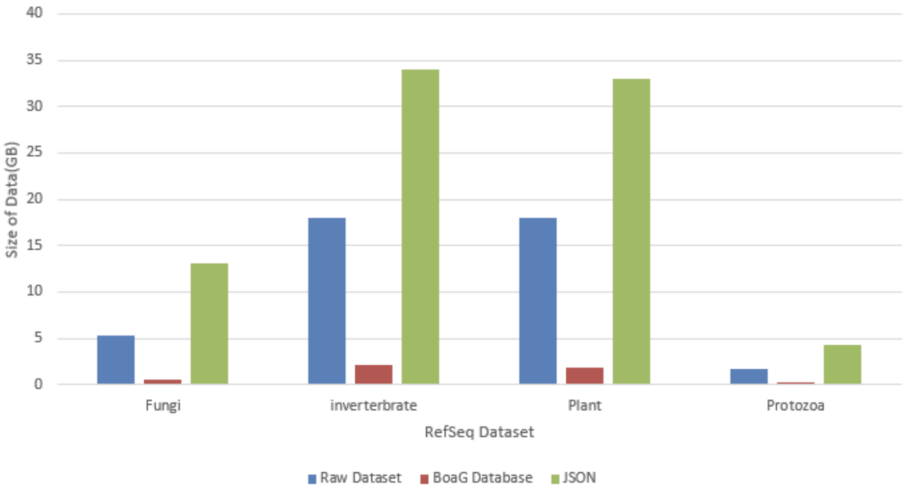


Figure 5.2 The Boa$_g$ database size comparison with the raw data in the RefSeq as well as the JSON version of the dataset.

5.3 Comparison between Python and Boa$_g$

A general-purpose language like Python could also be utilized to execute the same queries investigated here; however, the Python code would be larger and require learning how to use Python libraries. To illustrate, we wrote an example program in Python to calculate the top three most used assembly programs required only five lines of code in Boa$_g$ language. In Python, a similar analysis required 38 lines of code

```
var mapFunc1 = function(){
      emit (this.taxid, 1);};
var reduceFunc1 = function(key, values){
         return Array.sum(values); };
db.refseq1.mapReduce(
      mapFunc1,
      reduceFunc1,{out: "sum_taxid"}
    ).find()
```

(a) MongoDB

```
g: Genome = input;
counts: output sum[string] of int;
counts [g.taxid]<< 1;
```

(b) Boa$_g$

Figure 5.3 Comparison of the code needed to query the number of assembler programs per taxon id run on Refseq Data. On the left side, the MongoDB code needs eight lines of code in Python whereas the Boa$_g$ script needs only three lines of code.

Table 5.3 Comparison between MongoDB and Boa$_g$

Feature	**MongoDB**	**Boa$_g$**
Lines of Code	larger	smaller because it abstracts details of data analysis
Data generation time	longer due to the larger file	faster because of Binary file
Data file	JSON is 2.7 times larger than raw data	Hadoop Sequence file 5 times smaller than raw data
Schema Flexibility	Yes. Supports semi-structured data	Yes. Schema and compiler can be modified
MapReduce	Yes	Yes

(Figure 5.4). Because Python needs to aggregate the output data, it needs more lines of code and longer runtime. This advantage inherent to domain-specific languages will speed up a researcher's ability to query large datasets.

More comparisons in terms of runtime and lines of codes are given in Figure 5.5. These tests were performed on an iMac system with processor 4 GHz Intel Core i7 and 32 GB 1867 MHz DDR3 of memory.

An analysis in Boa$_g$ requires fewer lines of codes than other languages available like MongoDB and Python (Figure 5.3). The file size in the Boa$_g$ database is much smaller than the JSON file used in MongoDB, as Boa$_g$ utilizes a binary format. Since

Boa$_g$ also provides an external implementation that allows users to bring their implementation from Python, Perl, Bash, etc. Not all users of the infrastructure can run any arbitrary scripts on the infrastructure. Scripts need to be converted to a DSL function so that they will not cause security issues for the infrastructure.

Python

```
1 ... // imports
5 assembly_stats={}
6 def get_assembly(file):
7     assembly_program=None
8     with open(file, 'r') as f:
9     ... // retrieve assembly programs for each
    file
15      data=line.split(':')
16      assembly_program = data[1].strip()
25 ... // all assembly_stats.txt in the current
    directory
26 files_list=[f for f in os.listdir(".") if f.
    endswith('.txt')]
31 for f in files_list:
32  assembly_program=get_assembly(f)
33  if assembly_program in assembly_stats:
34   assembly_stats[assembly_program] +=1
35  else:
36   assembly_stats[assembly_program] = 1
37  sorted_assemblers = sorted(assembly_stats.
    items(), key=operator.itemgetter(1))
38  print(sorted_assemblers[-2:])
```

BoaG

```
1 g: Genome = input;
2 counts: output top(3) of string weight int;
3 assemblerData :=getAssembler(g.refseq);
4 foreach(i:int; def(assemblerData.assembler[i]))
5 counts << assemblerData.assembler[i].name weight
    1;
```

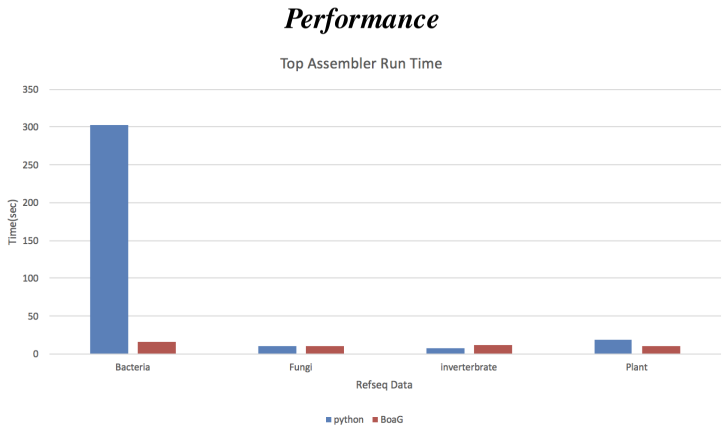


Figure 5.4 Comparison of Line of Code(LOC) and performance to answer query " What are the top three most used assembly programs?" run on Refseq Data. On the left side, the equivalent Boa$_g$ code needs 38 lines of code in Python whereas the Boa$_g$ script needs only five.

Task	Lines of Code(LOC)			Run Time (min)		
	Python	Boa$_g$	Diff	Python	Boa$_g$	Speedup
A. Summary Statistics across all species						
1. The average number of genome features in GFF files	39	7	4.2x	784	67	11x
2. Compute the mean and counts of feature size(base pairs)	33	7	4.7x	878	108	8x
3. Find the top 10 genomes with the largest and smallest genes	35	12	2.9x	1120	131	8x
B. Genome related questions						
1. Compute the number of reads for each genome	30	4	7.5x	620	20	31x
2. List of all tax ids and their counts	37	4	9.25x	54	1.2	45x
3. List the genomes within a specific genome size range	33	7	4.7x	62	1	62x
4. Find the smallest genome size within the clade	34	8	4.25x	55	1.5	36x
C. Feature related questions						
1. List of Genebank ID of all gene features in a specific range	32	15	2.1x	948	110	8.6x
2. The top genomes with smallest and largest mRNA length	30	11	2.72x	884	85	10x
3. What is the average mRNA length in each GFF?	27	9	3x	796	71	11x
D. Attribute related questions						
1. Gene, exon count, gene length, and average exons per gene	40	12	3.33x	1260	60	21x
2. Compute the shortest and largest CDS	44	12	3.66x	1124	75	15x
3. Compute the average number of mRNA per gene	45	11	4.09x	1320	65	20x
E. Assembler related questions						
1. What is the top used assembler within a clade?	32	5	6.4x	62	0.7	88x
2. The popularity of assembly programs over the years?	35	6	5.8x	60	0.7	85x
3. The most commonly used assembler in RefSeq	32	7	4.5x	50	0.5	100x
4. The best performing assembler in terms of contig N50?	31	6	5.1x	55	0.6	91x

Figure 5.5 Example of Boa$_g$ programs to compute different tasks on the full RefSeq dataset. The python programs were running on the single core. The Hadoop infrastructure on Bridges has 5 shared nodes with 32 mappers. While these queries can be written in parallel in python, this needs more lines of code and more programming skills to write a parallel code.

5.4 Conclusion

In this thesis, we presented Boa$_g$ which is a domain-specific language and shared data science infrastructure that takes advantage of Hadoop distribution for large-scale computations. Boa$_g$'s infrastructure opens the exploration of large datasets in ways that were previously not possible without deep expertise in data acquisition, data storage, data retrieval, data mining, and parallelization. The RefSeq database was used as an example dataset from Biology to show how to implement the domain-specific language Boa$_g$ for biological discovery. Boa$_g$ can query the RefSeq dataset in under 2 minutes for most queries, offering substantial time savings from other methods. Many examples, tutorials, and a Docker container are available at a GitHub repository. This thesis provides a proof of concept behind the Boa$_g$ infrastructure and its ability to scale to

much larger datasets. This is the first step towards providing a shared data science infrastructure to explore large biological datasets.

In the future, we will integrate new data types including the Non-Redundant protein database, biological ontologies, SRA, etc. We will also update the Boa$_g$ database and provide a publicly available web-interface for researchers to run queries on our infrastructure.

BIBLIOGRAPHY

[1] (2019). Generic feature format version 3. Accessed: 2019-02-07.

[2] (2019). Genomics england. Accessed: 2019-02-07.

[3] (2019). Hadoop and mongodb. Accessed: 2019-02-07.

[4] Alnasir, J. and Shanahan, H. (2018). The application of hadoop in structural bioinformatics. *BioRxiv*, page 376467.

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

[6] Bagheri, H., Muppirala, U., Masonbrink, R. E., Severin, A. J., and Rajan, H. (2019). Shared data science infrastructure for genomics data. *BMC bioinformatics*, 20(1):436.

[7] Biswas, S., Islam, M. J., Huang, Y., and Rajan, H. (2019). Boa meets python: a boa dataset of data science software in python language. In *Proceedings of the 16th International Conference on Mining Software Repositories*, pages 577–581. IEEE Press.

[8] Chodorow, K. (2013). *MongoDB: the definitive guide: powerful and scalable data storage*. " O'Reilly Media, Inc.".

[9] Cock, P. J., Antao, T., Chang, J. T., Chapman, B. A., Cox, C. J., Dalke, A., Friedberg, I., Hamelryck, T., Kauff, F., Wilczynski, B., et al. (2009). Biopython: freely available python tools for computational molecular biology and bioinformatics. *Bioinformatics*, 25(11):1422–1423.

[10] Dean, J. and Ghemawat, S. (2008). Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113.

[11] Dede, E., Govindaraju, M., Gunter, D., Canon, R. S., and Ramakrishnan, L. (2013). Performance evaluation of a mongodb and hadoop platform for scientific data analysis. In *Proceedings of the 4th ACM workshop on Scientific cloud computing*, pages 13–20. ACM.

[12] Deus, H. F., Correa, M. C., Stanislaus, R., Miragaia, M., Maass, W., De Lencastre, H., Fox, R., and Almeida, J. S. (2011). S3ql: A distributed domain specific language for controlled semantic integration of life sciences data. *BMC bioinformatics*, 12(1):285.

[13] Drost, H.-G. and Paszkowski, J. (2017). Biomartr: genomic data retrieval with r. *Bioinformatics*, 33(8):1216–1217.

[14] Dyer, R. (2013). Task fusion: Improving utilization of multi-user clusters. In *Proceedings of the 2013 companion publication for conference on Systems, programming, & applications: software for humanity*, pages 117–118. Citeseer.

[15] Dyer, R., Nguyen, H., Rajan, H., and Nguyen, T. (2015a). Boa: An enabling language and infrastructure for ultra-large-scale msr studies. In *The Art and Science of Analyzing Software Data*, pages 593–621. Elsevier.

[16] Dyer, R., Nguyen, H. A., Rajan, H., and Nguyen, T. N. (2013). Boa: A language and infrastructure for analyzing ultra-large-scale software repositories. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 422–431. IEEE Press.

[17] Dyer, R., Nguyen, H. A., Rajan, H., and Nguyen, T. N. (2015b). Boa: Ultra-large-scale software repository and source-code mining. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 25(1):7.

[18] Dyer, R., Rajan, H., Nguyen, H. A., and Nguyen, T. N. (2014a). Mining billions of ast nodes to study actual and potential usage of java language features. In *Proceedings of the 36th International Conference on Software Engineering*, pages 779–790. ACM.

[19] Dyer, R., Rajan, H., and Nguyen, T. N. (2014b). Declarative visitors to ease fine-grained source code mining with full history on billions of ast nodes. *ACM SIGPLAN Notices*, 49(3):23–32.

[20] Islam, M. J., Sharma, A., and Rajan, H. (2019). A cyberinfrastructure for big data transportation engineering. *Journal of Big Data Analytics in Transportation*.

[21] Koonin, E. V. and Wolf, Y. I. (2008). Genomics of bacteria and archaea: the emerging dynamic view of the prokaryotic world. *Nucleic acids research*, 36(21):6688–6719.

[22] Langmead, B., Hansen, K. D., and Leek, J. T. (2010). Cloud-scale RNA-sequencing differential expression analysis with Myrna. *Genome Biol.*, 11(8):R83.

[23] Leo, S., Santoni, F., and Zanetti, G. (2009). Biodoop: bioinformatics on hadoop. In *Parallel Processing Workshops, 2009. ICPPW'09. International Conference on*, pages 415–422. IEEE.

[24] Maddox, J., Long, Y., and Rajan, H. (2018). Large-scale study of substitutability in the presence of effects. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 528–538. ACM.

[25] Mahadik, K., Wright, C., Zhang, J., Kulkarni, M., Bagchi, S., and Chaterji, S. (2016). Sarvavid: A domain specific language for developing scalable computational genomics applications. In *Proceedings of the 2016 International Conference on Supercomputing*, ICS '16, pages 34:1–34:12, New York, NY, USA. ACM.

[26] Mernik, M., Heering, J., and Sloane, A. M. (2005). When and how to develop domain-specific languages. *ACM computing surveys (CSUR)*, 37(4):316–344.

[27] Niemenmaa, M., Kallio, A., Schumacher, A., KlemelÃd', P., Korpelainen, E., and Heljanko, K. (2012). Hadoop-bam: directly manipulating next generation sequencing data in the cloud. *Bioinformatics*, 28(6):876–877.

[28] Prlić, A., Yates, A., Bliven, S. E., Rose, P. W., Jacobsen, J., Troshin, P. V., Chapman, M., Gao, J., Koh, C. H., Foisy, S., et al. (2012). Biojava: an open-source framework for bioinformatics in 2012. *Bioinformatics*, 28(20):2693–2695.

[29] Pruitt, K. D., Tatusova, T., and Maglott, D. R. (2006). Ncbi reference sequences (refseq): a curated non-redundant sequence database of genomes, transcripts and proteins. *Nucleic acids research*, 35(suppl_1):D61–D65.

[30] Rajan, H. (2017). Bridging the digital divide in data science. *SPLASH-I: The ACM SIGPLAN conference on Systems, Programming, Languages and Applications: Software for Humanity (SPLASH)*.

[31] Sadasivam, G. S. and Baktavatchalam, G. (2010). A novel approach to multiple sequence alignment using hadoop data grids. In *Proceedings of the 2010 Workshop on Massive Data Analytics on the Cloud*, MDAC '10, pages 2:1–2:7, New York, NY, USA. ACM.

[32] Schmidt, B. and Hildebrandt, A. (2017). Next-generation sequencing: big data meets high performance computing. *Drug Discovery Today*.

[33] Smedley, D., Haider, S., Ballester, B., Holland, R., London, D., Thorisson, G., and Kasprzyk, A. (2009). Biomart–biological queries made easy. *BMC genomics*, 10(1):22.

[34] Stajich, J. E., Block, D., Boulez, K., Brenner, S. E., Chervitz, S. A., Dagdigian, C., Fuellen, G., Gilbert, J. G., Korf, I., Lapp, H., et al. (2002). The bioperl toolkit: Perl modules for the life sciences. *Genome research*, 12(10):1611–1618.

[35] Taylor, R. C. (2010). An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics. *BMC Bioinformatics*, 11 Suppl 12:S1.

[36] Terrizzano, I. G., Schwarz, P. M., Roth, M., and Colino, J. E. (2015). Data wrangling: The challenging yourney from the wild to the lake. In *CIDR*.

[37] Tiwari, N. M., Upadhyaya, G., Nguyen, H. A., and Rajan, H. (2017). Candoia: A platform for building and sharing mining software repositories tools as apps. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pages 53–63. IEEE.

[38] Turnbull, C., Scott, R. H., Thomas, E., Jones, L., Murugaesu, N., Pretty, F. B., Halai, D., Baple, E., Craig, C., Hamblin, A., et al. (2018). The 100000 genomes project: Bringing whole genome sequencing to the nhs. *BMJ: British Medical Journal (Online)*, 361.

[39] Upadhyaya, G. and Rajan, H. (2017). On accelerating ultra-large-scale mining. In *Proceedings of the 39th International Conference on Software Engineering: New Ideas and Emerging Results Track*, pages 39–42. IEEE Press.

[40] Upadhyaya, G. and Rajan, H. (2018a). Collective program analysis. In *Proceedings of the 40th International Conference on Software Engineering*, pages 620–631. ACM.

[41] Upadhyaya, G. and Rajan, H. (2018b). On accelerating source code analysis at massive scale. *IEEE Transactions on Software Engineering*, 44(7):669–688.

[42] Zhang, T., Upadhyaya, G., Reinhardt, A., Rajan, H., and Kim, M. (2018). Are code examples on an online Q&A forum reliable?: a study of API misuse on stack overflow. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pages 886–896. IEEE.