Shared Data Science Infrastructure for Genomics Data

Hamid Bagheri (hbagheri@iastate.edu)
Iowa State University https://orcid.org/0000-0003-1540-7797
Usha Muppirala
Iowa State University
Rick Masonbrink
Iowa State University
Andrew J Severin
Iowa State University
Hridesh Rajan
Iowa State University

Research article

Keywords: Shared Data Science Infrastructure, Domain-Specific Language, Boag, Genome Annotation

Posted Date: June 25th, 2019

DOI: https://doi.org/10.21203/rs.2.4295/v3

Version of Record: A version of this preprint was published on August 22nd, 2019. See the published version at https://doi.org/10.1186/s12859-019-2967-2.

Abstract

Background: 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 like Boa_g is needed to efficiently process and parse data contained in large data repositories. The main features of Boa_g are inspired from existing languages for data intensive computing and can easily integrate data from biological data repositories. Results: As a proof of concept, Boa for genomics, 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 speed up in its ability to query large datasets due to automated parallelization and distribution of Hadoop infrastructure during computations. Conclusions: 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 a greater access to researchers to efficiently explore data in new ways. We demonstrate the potential of a 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.

Background

As sequencing data continues to pile up in the online repositories [1], 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 programing 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 [2]. 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 [3]. 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). Running on a Hadoop cluster [4], 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 [4]. 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.

Domain specific languages and Databases in Bioinformatics

Genomics-specific languages are also common in high-throughput sequencing analysis such as S3QL, which aims to provide biological discovery by harnessing Linked Data [5]. In addition, there are libraries like BioJava [6], Bioperl [7], and Biopython [8] 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 [9] 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 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. There are organizations that have used the power of MongoDB and Hadoop framework together [10] to address challenges in Big Data. Genomics England [11] runs the 100,000 Genomes Project [12] 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 Hadoop and MapReduce programming model. Heavy computation applications like BLAST, GSEA and GRAMMAR have been implemented in Hadoop [13]. SARVAVID [14] has implemented five well-known applications for running on Haddop: BLAST, MUMmer, E-MEM, SPAdes, and SGA. BLAST [15] was also rewritten for Hadoop by Leo *et.al.* [16]. In addition to these programs, there are other efforts based on Hadoop to address RNA-Seq and sequence alignment [17, 18, 19].

A significant barrier to utilize the Hadoop framework in bioinformatics is the difficulty of the interface and the amount of expertise that are needed to write a MapReduce programs [20]. The proposed work tries to abstract away details of these complexities and open a door for more bioinformatics application. 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 $_g$, on the other hand is such a tool but is currently only implemented for mining very large software repositories like GitHub and Sourceforge. It recently has been applied to address potentials and challenges of Big Data in transportation [21].

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 potential 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 is currently computationally and logistically prohibitive for all but the most well-funded and staffed research groups. While BioMart [22], 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 [23] is an R package to retrieve raw genomics data that tries 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 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?

Results

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 species. Similarly, while each of these genomes have an annotation file, querying and summarizing information contained in this annotation file from several related genomes such as average number of genes, 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 is 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.

What is the largest and smallest genome in RefSeq? As of February 16th, 2019, the largest genome in the RefSeq database was Orycteropus afer 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 statistic only required seven lines of Boa_g code (Figure 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.

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, Ascomycota (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 RefSeq database (Table 1, Table 2). The number of genes, number of exons and exons per gene have increased for all clades database (Table 1, Table 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.

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 [24]. However, the mean gene length for these clades has not changed between timepoints, 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 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.

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).

This query required six lines of Boa_g code Figure 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.

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 3 and Table 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 required 10 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

Discussions

Database storage efficiency and computational efficiency with Hadoop

One benefit of the Boa$_g$ database is the significant reduction in 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 Hadoop Sequence file which makes disk writing faster than a text file (Figure 6). 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.

A 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 7, 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 are 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 minutes on assembly related queries. This lower bound 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.

Taking advantages of Hadoop based infrastructure, all the queries in the Table 5 and Table 6 that describe the genome assembly statistics before and after 2016 transition required less than a minute.

Comparison between MongoDB and Boa$_g$

An analysis in Boa$_g$ requires fewer lines of codes than other languages available like MongoDB and Python (Figure 9). 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 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 6). The JSON file size is

larger and on average it is more than double 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 9), thereby providing an easier interface for bioinformaticians to explore big data.

The performance of MongoDB and Hadoop has been previously compared [25], showing that the read-write overhead of Hadoop has a lower read-write overhead (Table 7).

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 (Figure 10). Because Python needs to aggregate the output data, it needs more lines of code and a 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 11. These tests were performed on an iMac system with processor 4 GHz Intel Core i7 and 32 GB 1867 MHz DDR3 of memory.

Boa$_g$ also provides an external implementation that allows users to bring their own 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.

Conclusion

In this work, 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$_g$ is able to query the RefSeq dataset in under 2 minutes for most queries, offering a substantial time savings from other methods. Many examples, tutorials, and a Docker container are available a GitHub repository. This paper 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 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 query on our infrastructure.

Methods

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 has 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.

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

• A computational framework on top of Hadoop that can query large dataset 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.

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 8, Genome, Sequence, Feature, and Assembler are types in Boa$_g$ language 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$_g$.

Output Aggregators in Boa$_g$

Table 9 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 [26], 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 also can contain indices that would be a grouping operation similar to traditional query languages.

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 8 (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). In order to integrate 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 with 5 nodes and up to 256 map tasks.

Data availability

All scripts, step by step process of scientific discovery, and additional examples of Boa queries used in this paper 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.

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/.

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 [27] 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.

Abbreviations

Boa*g* : Boa for Genomics; SDSI: Shared Data Science Infrastructure; DSL: Domain-Specific Language; RefSeq: Reference sequence database

Declarations

Ethics approval and consent to participate

Not applicable.

Consent for publication

Not applicable.

Availability of data and material

Boa*g* compiler's source code, documentation, Docker container, etc are provided on the GitHub repository (https://github.com/boalang/bio) The Boa*g* website is located here (https://boalang.github.io/bio/).

Competing interests

The authors declare that they have no competing interests.

Funding

This study was supported by the National Science Foundation under Grant CCF-15-18897 and CNS-15-13263 and the VPR office at Iowa State University. The listed funders played no role in the design of the study, data generation, implementation or in writing the manuscript.

Author's contributions

AJS and UM conceived of the application of Boa*g* to the RefSeq database and contributed to exploring the dataset from a biological perspective. HB wrote the codes, implemented the genomics specific types and customized compiler. He ran analysis and prepared figures. UM provided a first outline of the paper that HB, AJS, and RM later fleshed out before additional rounds of major editing by all authors. HR invented the notion of Shared Data Science Infrastructure (SDSI) [28], and the idea of SDSI for genomics data. He also contributed to the design of the Boa*g* domain-specific language for computing over data, and domain-specific types for representing RefSeq data. All the authors read and approved the final manuscript.

Acknowledgments

We would like to thank the Bridges supercomputing center for providing support with Hadoop.

Author details

1 Department of Computer Science, Iowa State University, 226 Atanasoff Hall, 50011 Ames,

US. 2 Genome Informatics Facility, Iowa State University, 206 Science I, 50011 Ames, US.

References

1. Schmidt, B., Hildebrandt, A.: Next-generation sequencing: big data meets high performance computing. Drug Discovery Today (2017)

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

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

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

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

6. Prlic´, 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.: Biojava: an open-source framework for bioinformatics in 2012. Bioinformatics 28(20), 2693–2695 (2012)

7. 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.: The bioperl toolkit: Perl modules for the life sciences. Genome research 12(10), 1611–1618 (2002)

8. 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.: Biopython: freely available python tools for computational molecular biology and bioinformatics. Bioinformatics 25(11), 1422–1423 (2009)

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

10. Hadoop and MongoDB. https://www.mongodb.com/hadoop-and-mongodb

11. Genomics England. https://www.genomicsengland.co.uk/

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

13. Taylor, R.C.: An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics. BMC Bioinformatics 11 Suppl 12, 1 (2010)

14. Mahadik, K., Wright, C., Zhang, J., Kulkarni, M., Bagchi, S., Chaterji, S.: Sarvavid: A domain specific language for developing scalable computational genomics applications. In: Proceedings of the 2016 International Conference on Supercomputing. ICS '16, pp. 34–13412. ACM, New York, NY, USA (2016). doi:10.1145/2925426.2926283. http://doi.acm.org/10.1145/2925426.2926283

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

16. Leo, S., Santoni, F., Zanetti, G.: Biodoop: bioinformatics on hadoop. In: Parallel Processing Workshops, 2009. ICPPW'09. International Conference On, pp. 415–422 (2009). IEEE

17. Niemenmaa, M., Kallio, A., Schumacher, A., Klemelä, P., Korpelainen, E., Heljanko, K.: Hadoop-bam: directly manipulating next generation sequencing data in the cloud. Bioinformatics 28(6), 876–877 (2012). doi:10.1093/bioinformatics/bts054

18. Sadasivam, G.S., Baktavatchalam, G.: 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, pp. 2–127. ACM, New York, NY, USA (2010). doi:10.1145/1779599.1779601. http://doi.acm.org/10.1145/1779599.1779601

19. Langmead, B., Hansen, K.D., Leek, J.T.: Cloud-scale RNA-sequencing differential expression analysis with Myrna. Genome Biol. 11(8), 83 (2010)

20. Alnasir, J., Shanahan, H.: The application of hadoop in structural bioinformatics. BioRxiv, 376467 (2018)

21. Islam, M.J., Sharma, A., Rajan, H.: A cyberinfrastructure for big data transportation engineering. Journal of Big Data Analytics in Transportation (2019). doi:10.1007/s42421-019-00006-8

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

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

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

25. Dede, E., Govindaraju, M., Gunter, D., Canon, R.S., Ramakrishnan, L.: Performance evaluation of a mongodb and hadoop platform for scientific data analysis. In: Proceedings of the 4th ACM Workshop on Scientific Cloud Computing, pp. 13–20 (2013). ACM

26. Chodorow, K.: MongoDB: the Definitive Guide: Powerful and Scalable Data Storage. " O'Reilly Media, Inc.", ??? (2013)

27. Pruitt, K.D., Tatusova, T., Maglott, D.R.: Ncbi reference sequences (refseq): a curated non-redundant sequence database of genomes, transcripts and proteins. Nucleic acids research 35(suppl_1), 61–65 (2006)

28. Rajan, H.: Bridging the digital divide in data science. In: SPLASH/SPLASH-I'17: The ACM SIGPLAN Conference on Systems, Programming, Languages and Applications: Software for Humanity (2017)

29. Generic Feature Format Version 3. http://gmod.org/wiki/GFF3

Tables

Due to technical limitations, the tables cannot be displayed in HTML. Please find the tables in the supplemental files below.

Figures

```
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 1

Code to find the smallest and largest genomes in RefSeq.

```
g: Genome = input;
geneCounts: output sum[string][string] of int;
exonCounts: output sum[string][string] of int;
adata := getAssembler(g.refseq);
asYear :=yearof(adata.assembly_date);
if(asYear >= 2016)
 foreach(i:int; def(g.sequence[i])){
  fdata:=getFeature(g.refseq,g.sequence[i].accession);
  foreach(j:int; def(fdata.feature[j])){
   if(match("gene",fdata.feature[j].ftype))
    geneCounts [g.refseq][g.taxid]<< 1;
   if(match("exon",fdata.feature[j].ftype))
    exonCounts [g.refseq][g.taxid]<< 1;
 }
}
```

Figure 2

Number of exons, genes, and exons per gene after 2016. The output is shown in Table 1.

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

Figure 3

Bacterial assembly programs popularity over time. The output of this script is shown in Figure 4.

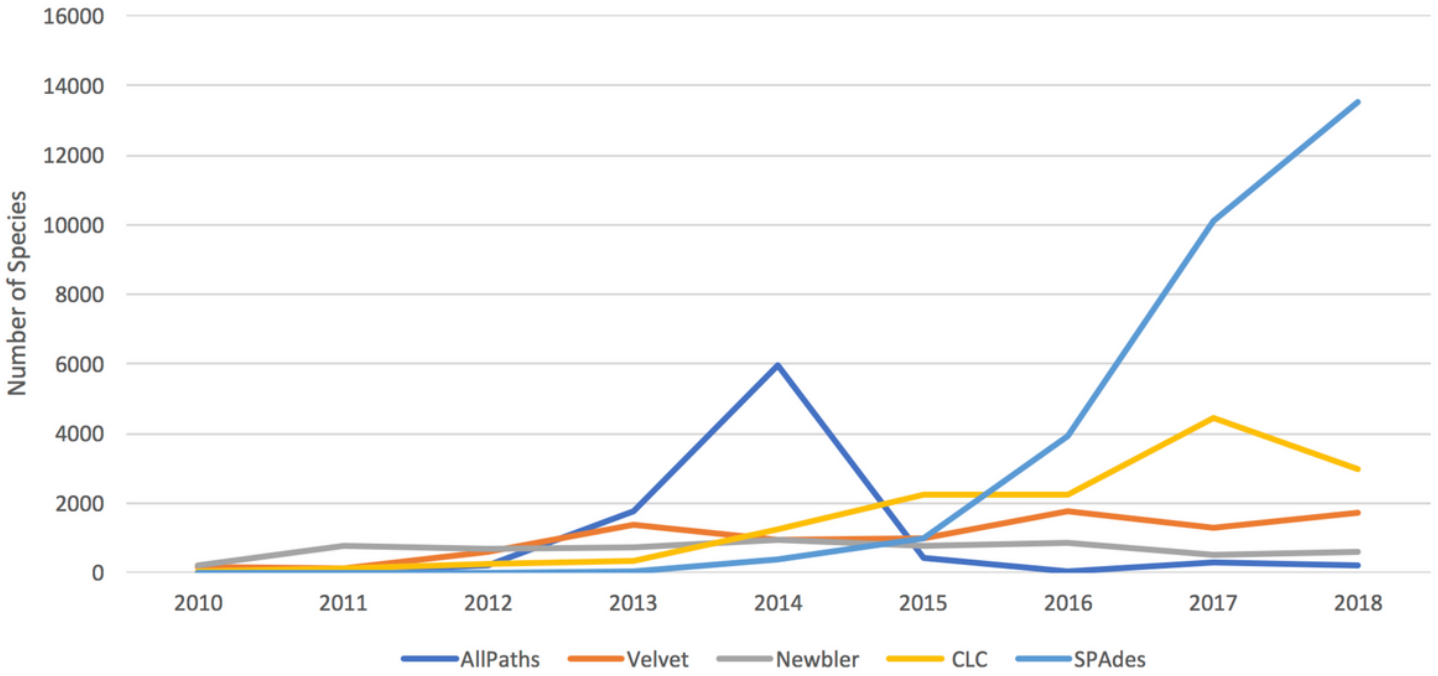

Figure 4

Assembler programs for Bacteria over the years

```
g: Genome = input;
counts: output collection [string][string]
[int][int][int][int][int][int] of int;
adata := getAssembler(g.refseq);
asYear :=yearof(adata.assembly_date);
if(asYear >= 2016)
  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 5

Assembly statistics for genomes for years after 2016. The output is shown in Table 5.

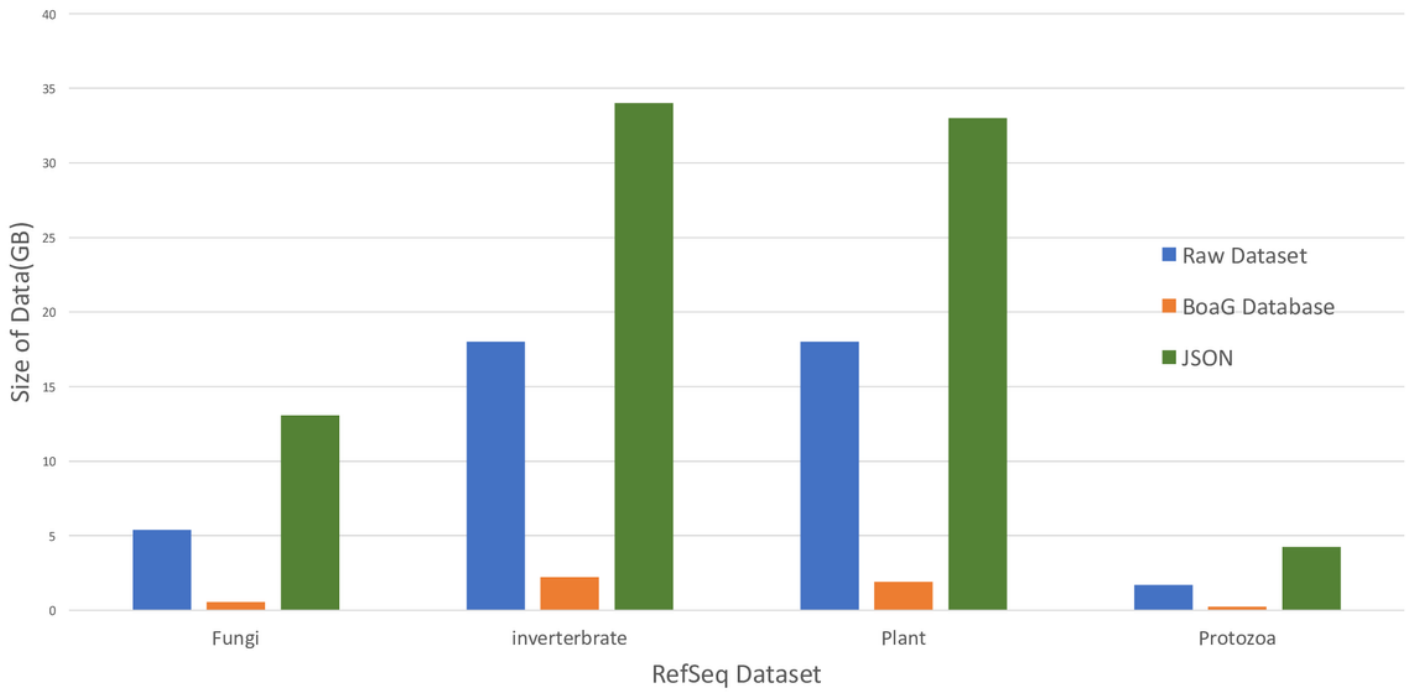

Figure 6

The Boag database size comparison with the raw data in the RefSeq as well as the JSON version of the dataset.

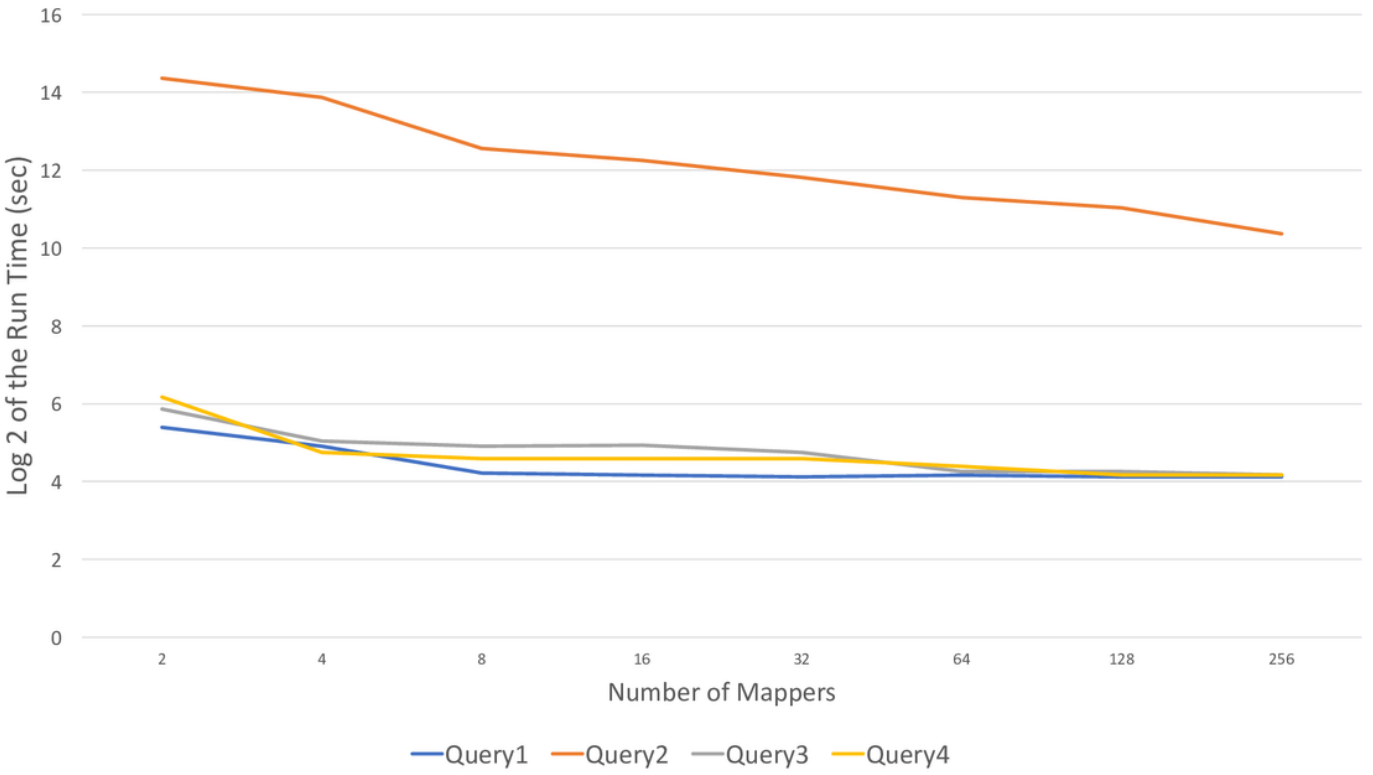

Figure 7

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

```
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) BoaG

Figure 8

Boag Architecture and Data Generation

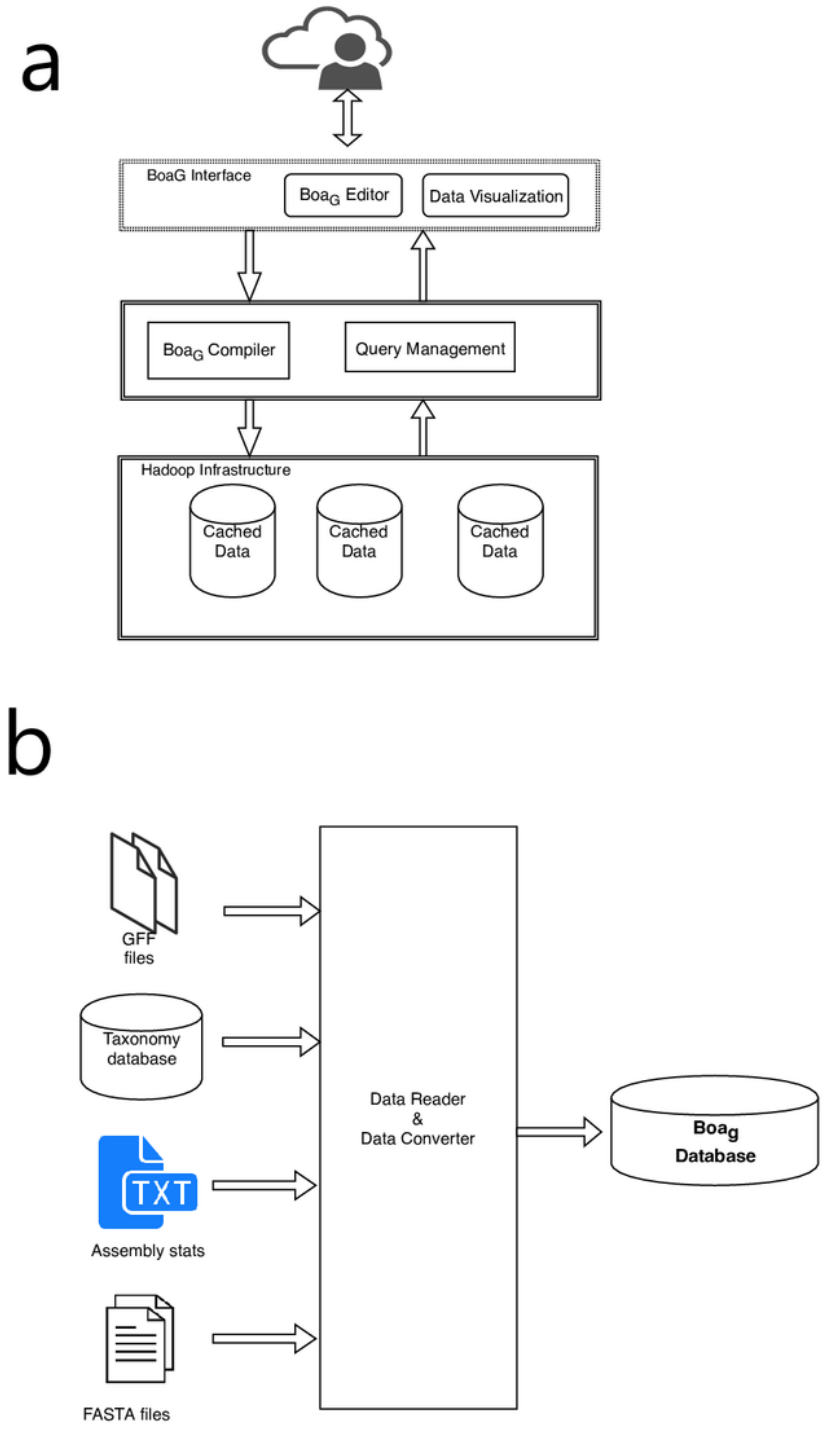

Figure 9

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 BoaG script needs only three lines of code.

Python

```
... // imports


assembly_stats={}
def get_assembly(file):
    assembly_program=None
    with open(file, 'r') as f:
    ... // retrieve assembly programs for each file


     data=line.split(':')
     assembly_program = data[1].strip()


... // all assembly_stats.txt in the current directory
files_list=[f for f in os.listdir(".") if f.endswith('.txt')]


for f in files_list:
 assembly_program=get_assembly(f)
 if assembly_program in assembly_stats:
  assembly_stats[assembly_program] +=1
 else:
  assembly_stats[assembly_program] = 1
 sorted_assemblers = sorted(assembly_stats.items(), key=operator.it
print(sorted_assemblers[-2:])
```

BoaG

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

Performance

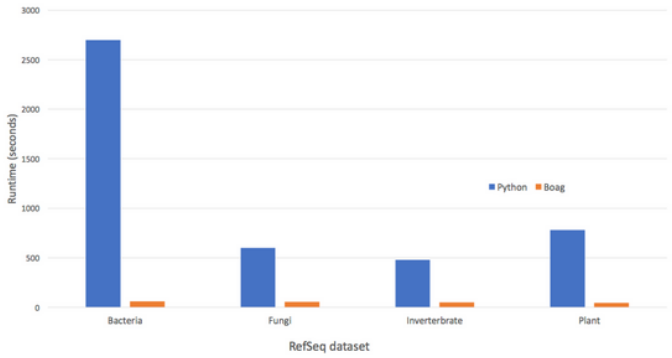

Figure 10

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 Boag code needs 38 lines of code in Python whereas the Boag 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. Compute 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 ten 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. Generate list of Genebank ID of all gene features in a specific range.	32	15	2.1x	948	110	8.6x
2. Find the top 10 genomes with the smallest and largest mRNA feature 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. Compute gene, exon count, gene length, and average number of 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. What is the most commonly used assembler across all genome sizes?	32	7	4.5x	50	0.5	100x
4. What is the best performing assembler in terms of contig N50?	31	6	5.1x	55	0.6	91x

Figure 11

Example of Boag 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.

Supplementary Files

This is a list of supplementary files associated with this preprint. Click to download.

- supplement1.pdf