frontiers

MG7: Configurable and scalable 16S metagenomics data analysis – new methods optimized for massive cloud computing

Alexey Alekhin [1] **Evdokim Kovach** [1] **Marina Manrique** [1] **Pablo Pareja** [1] **Eduardo Pareja** [1] **Raquel Tobes** [1] **and Eduardo Pareja-Tobes** [1,*]

[1] *Oh no sequences! Research Group, Era7 Bioinformatics, Granada, Spain*

Correspondence*:
Corresponding Author
Oh no sequences! Research Group, Era7 Bioinformatics, Plaza Campo Verde 3,
Granada, 18001, Spain, eparejatobes@ohnosequences.com

ABSTRACT

No abstract yet. Will be here.

Keywords: Metagenomics, 16S, Bacterial diversity profile, Bio4j, Graph databases, Cloud computing, NGS, Genomic big data

1 INTRODUCTION

Metagenomics data analysis is growing at exponential rate during the last years. The increasing throughput of massively parallel sequencing technologies, the derived decreasing cost, and the high impact of metagenomics studies, especially in human health (diagnostics, treatments, drug response, prevention), are crucial reasons responsible for this growth of Metagenomics. There is a growing interest in sequencing all kind of microbiomes (gut, mouth, skin, urinary tract, airway, milk, bladder), in different conditions of health and disease, or after different treatments. Metagenomics is also impacting environmental sciences, crop sciences, agrifood sector and biotechnology in general. This new possibilities for exploring the diversity of micro-organisms in the most diverse environments is opening many new research areas but, due to this wide interest, it is expected that the amount of data will be overwhelming in the short time (Stephens et al., 2015).

Genome researchers have raised the alarm over big data in the past nature news add ref but even a more serious challenge might be faced with the metagenomics boom/ upswing. If we compare metagenomics data with other genomics data used in clinical genotyping we find a differential feature: the key role of time. Thus, for example, in some longitudinal studies, serial sampling of the same patient along several weeks (or years) is being used for the follow up of some intestinal pathologies, for studying the evolution of gut microbiome after antibiotic treatment, or for colon cancer early detection (Zeller et al., 2014). This need of sampling across time adds more complexity to metagenomics data storage and demands adapted algorithms to detect state variations across time as well as idiosyncratic commonalities of the microbiome of each individual (Franzosa et al., 2015). In addition to the intra-individual sampling-time dependence, metagenomic clinical test results vary depending on the specific region of extraction of the clinical specimen. This local variability adds complexity to the analysis since different localizations (different tissues, different anatomical regions, healthy or tumour tissues) are required to have a sufficiently complete landscape of the human microbiome. Moreover, reanalysis of old samples using new tools and better reference databases might be also demanded from time to time.

During the last years other sciences as astronomy or particle physics are facing the big data challenge but, at least, these science have standards for data processing (Stephens et al., 2015). Global standards for converting raw sequence data into processed data are not yet well defined in metagenomics and there are shortcomings derived from the fact that many bioinformatics methodologies currently used for metagenomics data analysis were designed for a scenario very different that the current one. These are some of the aspects that have suffered crucial changes and advances with a direct impact in metagenomics data analysis:

i. The first aspect is related to the sequences to be analyzed: the reads are larger, the sequencing depth and the number of samples of each project are considerably bigger. The first metagenomics studies were very local projects, while nowadays the most fruitful studies are done at a global level (international, continental, national). This kind of global studies has yielded the discovery of clinical biomarkers for diseases of the importance of cancer, obesity or inflammatory bowel diseases and has allowed exploring the biodiversity in many earth environments

ii. The second aspect derives from the impressive genomics explosion, its effect being felt in this case in the reference sequences. The immense amount of sequences available in public repositories demands new approaches in curation, update and storage for metagenomics reference databases: current models will or already have problems to face the future avalanche of metagenomic sequences.

iii. The third aspect to consider for metagenomics data analysis is related to the appearance of new models for massive computation and storage such as the so-called cloud, or the widespread adoption of programming methodologies like functional programming, or, more speculatively, dependently typed programming. The immense new possibilities that these advances offer must have a direct impact in metagenomics data analysis.

iv. And finally the new social manner to do science, and especially genomic science is the fourth aspect to consider. Metagenomics evolves in a social and global scenario following a science democratization trend in which many small research groups from distant countries share a common big metagenomics project. This global cooperation demands systems allowing following exactly the same pipelines using equivalent cloud resources to modularly execute the analysis in an asynchronous way of working between different groups. This new scenario calls for new methods and tools to handle the current and future volume of metagenomic data with the sufficient speed of analysis.

Considering all these aspects we have designed a new open source methodology for analyzing metagenomics data that exploits the new possibilities that cloud computing offers to get a system robust, programmatically configurable, modular, distributed, flexible, scalable and traceable in which the biological databases of reference sequences can be easily updated and/or frequently substituted by new ones or by databases specifically designed for focused projects.

2 RESULTS

2.1 Overview

To tackle the challenges posed by metagenomics big data analysis outlined in the Introduction

- Static reproducible specification of dependencies and behavior of the different components using *Statika* and *Datasets*

- Parallelization and distributed analysis based on AWS, with on-demand infrastructure as the basic paradigm
- Definition of complex pipelines using *Loquat*, a composable system for scaling/parallelizing stateless computations especially designed for Amazon Web Services (AWS)
- A new approach to data analysis specification, management and specification based on working with it in exactly the same way as for a software project, together with the extensive use of compile-time structures and checks
- Modeling of the taxonomy tree using the new paradigm of graph databases (Bio4j). It facilitates the taxonomic assignment tasks and the calculation of the taxa abundance values considering the hierarchic structure of taxonomy tree (cumulative values)
- per-read assignment (??)

2.2 Libraries and resources

In this section we describe the resources and Scala libraries developed by the authors on top of which MG7 is built.

Scala, a hybrid object-functional programming language, was chosen based on the possibility of using certain advanced programming styles, and Java interoperability, which let us build on the vast number of existing Java libraries; we take advantage of this when using Bio4j as an API for the NCBI taxonomy. It has support for type-level programming, type-dependent types (through type members) and singleton types, which permits a restricted form of dependent types where types can depend essentially on values determined at compile time (through their corresponding singleton types). Conversely, through implicits one can retrieve the value corresponding to a singleton type.

2.2.1 *Statika*: machine configuration and behavior

Statika is a Scala library developed by the first and last authors which serves as a way of defining and composing machine behaviors statically. The main component are **bundles**. Each bundle declares a sequence of computations (its behavior) which will be executed in an **environment**. A bundle can *depend* on other bundles, and when being executed by an environment, its DAG of dependencies is linearized and run in sequence. In our use, bundles correspond to what an EC2 instance should do and an environment to an image (AMI: **A**mazon **M**achine **I**mage) which prepares the basic configuration, downloads the Scala code and runs it.

2.2.2 *Datasets*: a mini-language for data

Datasets is a Scala library developed by the first and last authors with the goal of being a Scala-embedded mini-language for datasets and their locations. **Data** is represented as type-indexed fields: Keys are modeled as singleton types, and values correspond to what could be called a denotation of the key: a value of type `Location` tagged with the key type. Then a **Dataset** is essentially a collection of data, which are guaranteed statically to be different through type-level predicates, making use of the value type correspondence which can be established through singleton types and implicits. A dataset location is then just a list of locations formed by locations of each data member of that dataset. All this is based on what could be described as an embedding in Scala of an extensible record system with concatenation on disjoint labels, in the spirit of (Harper and Pierce, 1990, Harper and Pierce (1991)). For that *Datasets* uses ohnosequences/cosas.

Data keys can further have a reference to a **data type**, which, as the name hints at, can help in providing information about the type of data we are working with. For example, when declaring Illumina reads as a data, a data type containing information about the read length, insert size or end type (single or paired) is used.

A **location** can be, for example, an S3 object or a local file; by leaving the location type used to denote particular data free we can work with different "physical" representations, while keeping track of to which logical data they are a representation of. Thus, a process can generate locally a `.fastq` file representing the merged reads, while another can put it in S3 with the fact that they all correspond to the "same" merged reads is always present, as the data that those "physical" representations denote.

2.2.3 *Loquat*: Parallel data processing with AWS

Loquat is a library developed by the first, second and last authors designed for the execution of embarrassingly parallel tasks using S3, SQS and EC2.

A **loquat** executes a process with explicit input and output datasets (declared using the *Datasets* library described above). Workers (EC2 instances) read from an SQS queue the S3 locations for both input and output data; then they download the input to local files, and pass these file locations to the process to be executed. The output is then put in the corresponding S3 locations.

A manager instance is used to monitor workers, provide initial data to be put in the SQS queue and optionally release resources depending on a set of configurable conditions.

Both worker and manager instances are Statika bundles. In the case of the worker, it can declare any dependencies needed to perform its task: other tools, libraries, or data.

All configuration such as the number of workers or the instance types is declared statically, the specification of a loquat being ultimately a Scala object. There are deploy and resource management methods, making it easy to use an existing loquat either as a library or from (for example) a Scala REPL.

The input and output (and their locations) being defined statically has several critical advantages. First, composing different loquats is easy and safe; just use the output types and locations of the first one as input for the second one. Second, data and their types help in not mixing different resources when implementing a process, while serving as a safe and convenient mechanism for writing generic processing tasks. For example, merging paired-end Illumina reads generically is easy as the data type includes the relevant information (insert size, read length, etc) to pass to a tool such as FLASH.

2.2.4 Type-safe eDSLs for BLAST and FLASH

We developed our own Scala-based type-safe eDSLs (**e**mbedded **D**omain **S**pecific **L**anguage) for FLASH and BLAST expressions and their execution.

In the case of BLAST we use a model for expressions where we can guarantee for each BLAST command expression at compile time

- all required arguments are provided
- only valid options are provided
- correct types for each option value
- valid output record specification

Generic type-safe parsers returning an heterogeneous record of BLAST output fields are also available, together with output data defined using *Datasets* which have a reference to the exact BLAST command options which yielded that output. This let us provide generic parsers for BLAST output which are guaranteed to be correct, for example.

In the same spirit as for BLAST, we implemented a type-safe EDSL for FLASH expressions and their execution, sporting features equivalent to those outlined for the BLAST EDSL.

2.2.5 Bio4j and Graph Databases

(Bio4j Pareja-Tobes et al., 2015) is a data platform integrating data from different resources such as UniProt or GO in a graph data paradigm. In the assignment phase we use a subgraph containing the NCBI Taxonomy, wrapping in Scala its Java API in a tree algebraic data type.

2.2.6 16S Reference Database Construction

Our 16S Reference Database is a curated subset of sequences from NCBI nucleotide database **nt**. The sequences included were selected by similarity with the bacterial and archaeal reference sequences downloaded from the **RDP database** (Cole et al., 2013). RDP unaligned sequences were used to capture new 16S sequences from **nt** using BLAST similarity search strategies and then, performing additional curation steps to remove sequences with poor taxonomic assignments to taxonomic nodes close to the root of the taxonomy tree. All the nucleotide sequences included in **nt** database has a taxonomic assignment provided by the **Genbank** sequence submitter. NCBI provides a table (available at ftp://ftp.ncbi.nlm.nih.gov/pub/taxonomy/) to do the mapping of any Genbank Identifier (GI) to its Taxonomy Identifier (TaxID). Thus, we are based on a crowdsourced submitter-maintained taxonomic annotation system for reference sequences. It supposes a sustainable system able to face the expected number of reference sequences that will populate the public global nucleotide databases in the near future. Another advantageous point is that we are based on NCBI taxonomy, the *de facto* standard taxonomic classification for biomolecular data (Cochrane and Galperin, 2010). NCBI taxonomy is, undoubtedly, the most used taxonomy all over the world and the most similar to the official taxonomies of each specific field. This is a crucial point because all the type-culture and tissue databanks follow this official taxonomical classification and, in addition, all the knowledge accumulated during last decades is referred to this taxonomy. In addition NCBI provides a direct connection between taxonomical formal names and the physical specimens that serve as exemplars for the species (Federhen, 2014).

Certainly, if metagenomics results are easily integrated with the theoretical and experimental knowledge of each specific area, the impact of metagenomics will be higher that if metagenomics progresses as a disconnected research branch. Considering that metagenomics data interoperability, which is especially critical in clinical environments, requires a stable taxonomy to be used as reference, we decided to rely on the most widely used taxonomy: the NCBI taxonomy. In addition, the biggest global sequence database GenBank follows this taxonomy to register the origin of all their submitted sequences. Our 16S database building strategy allows the substitution of the 16S database by any other subset of **nt**, even by the complete **nt** database if it would be needed, for example, for analyzing shotgun metagenomics data. This possibility of changing the reference database provides flexibility to the system enabling it for easy updating and project-driven personalization.

2.3 Pipeline Description

2.3.1 Taxonomic Assignment Algorithms

2.3.1.1 Lowest Common Ancestor based Taxonomic Assignment

For each read:

1. Select only one BLASTN alignment (HSP) per reference sequence (the HSP with lowest e value) 2. Filter all the HSPs with bitscore below a defined BLASTN bitscore threshold s_0 3. Find the best bitscore

value S in the set of BLASTN HSPs corresponding to hits of that read 4. Filter all the alignments with bitscore below p * S (where p is a fixed by the user coefficient to define the bitscore required, e.g. if p=0.9 and S=700 the required bitscore threshold would be 630) 5. Select all the taxonomic nodes to which map the reference sequences involved in the selected HSPs: - If all the selected taxonomic nodes forms a line in the taxonomy tree (are located in a not branched lineage to the tree root) we should choose the most specific taxID as the final assignment for that read - If not, we should search for the (sensu stricto) Lowest Common Ancestor (LCA) of all the selected taxonomic nodes (See Figure X)

In this approach the value used for evaluating the similarity is the bitscore that is a value that increases when similarity is higher and depends a lot on the length of the HSP

2.3.1.2 Best BLAST hit taxonomic assignment

We have maintained the simpler method of Best BLAST Hit (BBH) taxonomic assignment because, in some cases, it can provide information about the sequences that can be more useful than the obtained using LCA algorithm. Using LCA algorithm when some reference sequences with BLAST alignments over the required thresholds map to a not sufficiently specific taxID, the read can be assigned to an unspecific taxon near to the root. If the BBH reference sequence maps to a more specific taxa this method, in that case, gives us useful information.

2.4 Using MG7 with some example data-sets

We selected the datasets described in [Kennedy-2014] (??)

2.5 Availability

MG7 is open source, available at https://github.com/ohnosequences/mg7 under an AGPLv3 license.

3 DISCUSSION

3.1 What MG7 brings

We could summarize the most innovative ideas and developments in MG7:

1. Treat data analysis as a software project. This makes for radical improvements in *reproducibility*, *reuse*, *versioning*, *safety*, *automation* and *expressiveness*
2. input and output data, their locations and type are expressible and checked at compile-time using *Datasets* 3. management of dependencies and machine configurations using *Statika*
3. automation of AWS cloud resources and processes, including distribution and parallelization through the use of *Loquat*
4. taxonomic data and related operations are treated natively as what they are: graphs, through the use of *Bio4j*
5. MG7 provides a sustainable model for taxonomic assignment, appropriate to face the challenging amount of data that high throughput sequencing technologies generate

We will expand on each item in the following sections.

3.2 A new approach to data analysis

MG7 proposes to define and work with a particular data analysis task as a software project, using Scala. The idea is that *everything*: data description, their location, configuration parameters, the infrastructure used, . . . should be expressed as Scala code, and treated in the same way as any (well-managed) software project. This includes, among other things, using version control systems (`git` in our case), writing tests, making stable releases following semantic versioning or publishing artifacts to a repository.

What we see as key advantages of this approach (when coupled with compile-time specification and checking), are

- **Reproducibility** the same analysis can be run again with exactly the same configuration in a trivial way.
- **Versioning** as in any software project, there can be different versions, stable releases, etc.
- **Reuse** we can build standard configurations on top of this and reuse them for subsequent data analysis. A particular data analysis *task* can be used as a *library* in further analysis.
- **Decoupling** We can start working on the analysis specification, without any need for available data in a much easier way.
- **Documentation** We can take advantage of all the effort put into software documentation tools and practices, such as in our case Scaladoc or literate programming. As documentation, analysis processes and data specification live together in the files, it is much easier to keep coherence between them.
- **Expresiveness and safety** For example in our case we can choose only from valid Illumina read types, and then build a default FLASH command based on that. The output locations, being declared statically, are also available for use in further analysis.

3.3 Inputs, outputs, data: compile-time, expressive, composable

3.4 Tools, data, dependencies and machine configurations

3.5 Parallel cloud execution ??

3.6 Taxonomy and Bio4j

The hierarchic structure of the taxonomy of the living organisms is a tree, and, hence, is also a graph in which each node, with the exception of the root node, has a unique parent node. It led us to model the taxonomy tree as a graph using the graph database paradigm. Previously we developed Bio4j **[Pareja-Tobes-2015]**, a platform for the integration of semantically rich biological data using typed graph models. It integrates most publicly available data linked with sequences into a set of interdependent graphs to be used for bioinformatics analysis and especially for biological data.

3.7 Future-proof

3.8 MG7 Future developments

3.8.1 Shotgun metagenomics

It is certainly possible to adapt MG7 to work with shotgun metagenomics data. Simply changing the reference database to include whole genome sequence data could yield interesting results. This could also be refined by restricting reference sequences according to all sort of criteria, like biological function or taxonomy. Bio4j would be an invaluable tool here, thanks to its ability to express express complex predicates on sequences using all the information linked with them (GO annotations, UniProt data, NCBI taxonomy, etc).

3.8.2 Comparison of groups of samples

3.8.3 Interactive visualizations using the output files of MG7 (Biographika project)

4 MATERIALS AND METHODS

4.1 Amazon Web Services

We use EC2, S3 and SQS through a Scala wrapper of the official AWS Java SDK, ohnosequences/aws-scala-tools 0.13.2. This uses version `1.10.9` of the AWS Java SDK.

4.2 Scala

MG7 itself and all the libraries used are written in Scala `2.11`.

4.3 Statika

MG7 uses ohnosequences/statika 2.0.0 for specifying the configuration and behavior of EC2 instances.

4.4 Datasets

MG7 uses ohnosequences/datasets 0.2.0 for specifying input and output data, their type and their location.

4.5 Loquat

MG7 uses ohnosequences/loquat 2.0.0 for the specification of data processing tasks and their execution using AWS resources.

4.6 BLAST eDSL

MG7 uses ohnosequences/blast 0.2.0. The BLAST version used is `2.2.31+`

4.7 FLASH eDSL

MG7 uses ohnosequences/flash 0.1.0. The FLASH version used is `?.?.?`

4.8 Bio4j

MG7 uses bio4j/bio4j 0.12.0-RC3 and bio4j/bio4j-titan 0.4.0-RC2 as an API for the NCBI taxonomy.

5 ACKNOWLEDGEMENTS

The two first authors are funded by INTERCROSSING (Grant 289974).

REFERENCES

Cochrane, G. R. and Galperin, M. Y. (2010). The 2010 nucleic acids research database issue and online database collection: a community of data resources. *Nucleic acids research* 38, D1–D4

Cole, J. R., Wang, Q., Fish, J. A., Chai, B., McGarrell, D. M., Sun, Y., et al. (2013). Ribosomal database project: data and tools for high throughput rrna analysis. *Nucleic acids research* , gkt1244

Federhen, S. (2014). Type material in the ncbi taxonomy database. *Nucleic acids research* , gku1127

Franzosa, E. A., Huang, K., Meadow, J. F., Gevers, D., Lemon, K. P., Bohannan, B. J., et al. (2015). Identifying personal microbiomes using metagenomic codes. *Proceedings of the National Academy of Sciences* , 201423854

Harper, R. and Pierce, B. (1991). A record calculus based on symmetric concatenation. In *Proceedings of the 18th ACM SIGPLAN-SIGACT symposium on Principles of programming languages* (ACM), 131–142

Harper, R. W. and Pierce, B. C. (1990). Extensible records without subsumption

Pareja-Tobes, P., Tobes, R., Manrique, M., Pareja, E., and Pareja-Tobes, E. (2015). Bio4j: a high-performance cloud-enabled graph-based data platform. *bioRxiv* , 016758

Stephens, Z. D., Lee, S. Y., Faghri, F., Campbell, R. H., Zhai, C., Efron, M. J., et al. (2015). Big data: Astronomical or genomical? *PLoS Biol* 13, e1002195

Zeller, G., Tap, J., Voigt, A. Y., Sunagawa, S., Kultima, J. R., Costea, P. I., et al. (2014). Potential of fecal microbiota for early-stage detection of colorectal cancer. *Molecular systems biology* 10, 766