Multiscale Complex Genomics

Project Acronym: MuG
Project title: Multi-Scale Complex Genomics (MuG)
Call: H2020-EINFRA-2015-1
Topic: EINFRA-9-2015
Project Number: 676556
Project Coordinator: Institute for Research in Biomedicine (IRB Barcelona)
Project start date: 1/11/2015
Duration: 36 months

Deliverable 4.4: Database and ETL design and implementation

Lead beneficiary: The European Bioinformatics Institute (EMBL-EBI)
Dissemination level: PUBLIC

Due date: 15/04/2017
Actual submission date: 15/04/2017

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 676556.

Document History

Version	Contributor(s)	Partner	Date	Comments
0.1	Mark McDowall	EMBL-EBI	24/03/2017	First draft
0.2	Andrew Yates	EMBL-EBI	04/04/2017	Second draft
0.3	Mark McDowall	EMBL-EBI	06/04/2017	Read through and corrections
0.4	Laia Codo	BSC	11/04/2017	Physical storage and user project organisation
0.5	Josep Gelpi	BSC	12/04/2017	
0.6	Mark McDowall	EMBL-EBI	13/04/2017	Third draft
1.0	Andrew Yates	EMBL-EBI	13/04/2017	Release Candidate
1.1	Adam Hospital	IRB	22/04/2017	Modifications to MD/CG dynamics
1.2	David Castillo	CNAG-CRG	24/04/2017	Corrections regarding TADbit

Table of Contents

1 EXECUTIVE SUMMARY

The following document describes the mechanisms that have been developed for storing data within the MuG Virtual Research Environment (VRE) and the tracking of files that have been loaded for use in, or generated by, the pipelines described in D4.3. The storage is a mixture of NoSQL and indexed files allowing for optimal retrieval of information as part of the dissemination work that is taking place as part of D4.5. Where there has been a requirement, APIs and new pipelines have been developed to allow easy extraction and loading of information within the COMPSs architecture and as part of a RESTful interface. It is not expected for this storage to remain static, as requirements develop so should the API and the underlying data model.

2 INTRODUCTION

There are a large number of experimental tools that are required to resolve the architecture of the chromosomes within the nucleus [vanSteensel2010]. This also means that there are a large number of input and output file types that need to be tracked, stored and eventually made available for visualisation or distribution. It is important that there is an infrastructure to account for all files within the system so that when they are downloaded by the user they are able to describe how that file was generated. Likewise, the information from these files will need to be extracted and indexed so that the results can be requested in a logical and timely manner within the context of a RESTful framework.

Here we present several levels to handle the storage requirements within the MuG VRE. Each solution is tuned to the data attributes and indexing strategy required. The first level is the data management API which tracks the files that have been loaded into the VRE, where the files are physically located and in the case of results files a list of source files that were required. There is also matching metadata to describe the processes that were involved in generating the data. The second level is related to the modelling of the data within the VRE so that it can get used by other pipelines. The third level is to process the data so that it can be efficiently used as part of a RESTful interface allowing the visualisation of the data by other tools.

3 DATA STORAGE BACKGROUND

Traditionally databases have been tabular in format with defined columns and many rows describing entries. There are then links between entries in one table and those in another that have a given relationship. This works really well for a lot of data types and scenarios, but there are times when data does not easily fit into a relational database schema. NoSQL databases refer to a class of database that are either one or all of the follow: “non-SQL”, “non-relational” or “not just SQL”. There are a handful of broad classes of NoSQL databases that exist:

- Wide column stores (eg Cassandra, Vertica)
- Document stores (eg MongoDB, Elastic)
- Key-value (eg MemcacheDB)
- Triple stores (eg AllegroGraph, Virtuoso)
- Graph (eg neo4J, Titan)

There are also databases that mix two or more of these concepts, including relational databases (eg OrientDB - http://orientdb.com). Each has a specialised model for representing the data in a way that is more a fitting for the data type. This has the distinct benefit that the data type natively fits to the way it is modelled within the storage layer. Search algorithms can therefore be more efficient when it comes to querying that data. NoSQL options can also allow for larger, or more complex, datasets to be stored. In the case of document stores it removes the need to define a table(s) and so the data model can evolve along with the data type.

This flexibility means that there can be reductions in development, loading and/or search times. These savings require there has to be extra management layers to handle data that might not have all the required fields, increased complexity in the definition of relationships, or limitations in the way that data can be queried.

As the NoSQL options have been growing so have the ways of storing data within a traditional relational database. There has been the introduction of fractal trees that allow for rapid indexing at high insert speeds (TokuDB - https://www.percona.com/software/mysql-database/percona-tokudb) and R-Trees [Guttman1984] allowing for indexing over geometries and rapid querying geometric position. These enhancements when combined with the flexibility of relational data structures and improvements to their underlying algorithms ensure relational databases continue to be a viable storage layer option for many applications.

File based storage options have been extensively used in genomics for the rendering of tracks in genome browsers and take advantage of the 1D nature of genomic annotation. Most notable are the BAM format (http://samtools.github.io/hts-specs/SAMv1.pdf), BigBed and BigWig [Kent2010] formats and the Tabix [Li2011] format. These formats normally work by having input data sorted by chromosome and start position. The input data is then compressed into chunks and index the position of the chunks. These indexes are optimised for data retrieval by genomic location. There are additional file formats that allow for the storage of large multidimensional arrays where it is possible to query the data structure by the slicing of arrays. Formats that are currently being developed and used with the scientific communities include HDF5 (https://www.hdfgroup.org/), ASDF (https://github.com/spacetelescope/asdf) and Blaze (http://blaze.pydata.org/). This means that file based indexes are ideally suited to range based queries. File based indexes require wrappers to be written for loading and retrieval, but they are often smaller and it is easy to distribute the files when it is deployed on RESTful web servers.

Molecular Dynamics (MD) trajectories storing 3D coordinates are usually large files, in the order of ~GB, that are saved together with a topology file including information about the molecule (see Annex section 8.2). Usually long trajectories are split in different files, to avoid having single huge files that would be difficult to process in traditional computers. In the few existing MD databases, different approaches have been used to store MD information. The older procedures (e.g. MoDEL, Meyer et al., Structure 2010, 18, 1399-1409), tend to store trajectories in usual file systems (RAID storage systems), and metadata (topology) in relational databases, whereas the newer approximations (e.g. BigNASim [Hospital2016]) are already exploiting the power of NoSQL databases presented before, storing metadata and analysis in document-oriented stores (e.g. MongoDB), and cartesian coordinates in wide column stores (e.g. CassandraDB). Using wide column stores allows the retrieval, in a very fast and efficient way, of trajectory coordinates corresponding to a certain slice of time, and also trajectory coordinates for a certain set of atoms from the molecule. And using document-oriented stores offers the flexibility required to store new analysis without the necessity to modify the already existing schema.

4 FILE MANAGEMENT

Within the VRE there is the loading, transfer, transformation and generation of a large number of files. These files have sources, either external or internally based on other files within the system. Tracking these files, who they belong to, the provenance and the way that the file was generated is essential so the user is able to accurately describe how the data was generated during publication and submission to archives. The DM API (https://github.com/Multiscale-Genomics/mg-dm-api) was written to help solve this issue.

The DM API is a data management API written to handle the tracking of files within the VRE along with relevant metadata such that there are accurate records of how that information was generated. The API is built to make it easy for developers to access the files that are required for running pipelines without having to manage fixed file hierarchies within the filesystem. This should hopefully avoid conflicts where two files have the same name and location.

4.1 Data Structure

For each file there is a set number of required parameters (Table 4.1). Extra parameters can be defined to describe additional metadata. As a result the backend is ideally suited to a document store style database. MongoDB (https://www.mongodb.com) was selected due to its maturity, schema flexibility and prior experience of the database within the consortium. MongoDB's flexibility is essential when describing the metadata associated with each file. Our format allows for user-dependent expansion of certain fields as and when required.

Parameter	Required	Description
user_id	YES	The unique user ID for who the file is associated with
file_id		This is an auto-generated ID that is created when the data is entered. The ID is unique to the file and the user.
file_path	YES	Location of the file either within the file system or a URL to an archive or repository.
file_type	YES	File format. The current accepted file types are described in section 5.
data_type		This describes the type of data that is in the file. This is helpful as there are several formats that can have different data. For example FASTQ data can be related to RNA-seq, MNase-Seq, ChIP-seq, WGBS, etc.
taxon_id	YES	The taxonomic ID of the species from which the sample data was taken.
compressed		Whether the file has been compressed. Type of compression used depends on the format in question.
source_id		List of the file IDs that were used during the creation of this file.
meta_data	DEPENDENT	There are cases where additional data is required for some files that is not relevant to other file types. Files that have been generated and are dependent on alignments require that the meta_data has an 'assembly' key with the assembly for which the alignment was made against

creation_time		This is the time inserted by the API and is not required from the user.

Table 4.1: *List of parameters stored by the data management API and whether these parameters are required*

4.2 Validation

As part of the loading there is also a validation step. This is to ensure that the files that are loaded have the required fields. This step also checks that if the files have particular data types then they have the relevant matching metadata. In the case of the alignment file type BAM it is important to also know the assembly accession of the genome that it was aligned against. It is important to know which assembly as changes are frequently made, so retrieving alignments from a BAM file against the wrong version of the assembly can have detrimental effects on the interpretation of the results.

4.3 Data Access

Access to the files is provided via a Python API. This allows for the querying of files for a given user by file_id, file_type or taxon_id and allows for easily listing relevant files. Our API has been developed so that a RESTful interface can be placed over the top for the retrieval and querying of files for external services. Full documentation of installation and each of the methods can be found on ReadTheDocs at http://mg-dm-api.readthedocs.io/. ReadTheDocs automatically pulls any updates from GitHub and updates the relevant sections of the documentation if changes have been made.

4.4 File Formats

There are a set number of file types that are used and/or generated by the pipelines within the VRE, these have been listed below. Further descriptions about specific formats can be found in the Annex in section 8.2

- FASTA
- FASTQ
- Bed / BigBed
- Wig / BigWig
- SAM / BAM
- GEM / bt2 / bwa
- GFF3
- CGmap / ATCGmap
- PDB
- xtc / NetCDF / mdcrd
- top / tpr / parmtop
- rst / cpt
- HDF5
- lif / tiff / png
- RData
- JSON
- txt
- tsv
- tar

Many of the file formats are specific to one data type, others are applicable to multiple data types and pipelines hence the importance of the data type parameters to help describe the use of the file. Files such as HDF5, JSON, tsv and txt are important formats for storing structured data that is dependent on the pipeline that has generated it. The tar format is also used for the storage of projects within the description of the VRE. This allows for multiples of files to get collated in a single place when maintaining the hierarchy of a working directory.

5 FILE STORAGE AND INDEXING

There are many files that need to be tracked within the VRE all representing different stages along the timeline from primary information loaded by the user to final results files. Within the results files there are files that represent tracks to be loaded into genome browsers, 2D arrays of data or 3D models and images. All need to be readily available so that they can be requested efficiently. In some cases calling the whole file, or range of files, can be prohibitively expensive due to number or size, so it is best to request just the information that is required. Being able to easily and quickly slice the data without being too CPU or RAM intensive is essential for a responsive web data interface.

Of the files that are described in section 4.4 and Annex 8.2 there are several groups of files that have different characteristics when it comes to data retrieval. There are images, where the relevant set of images are returned on a per project basis. The second group relate to sequence aligned data where the majority of requests revolve around searching for features that are present within a defined region on a chromosome. The final set are modelling datasets containing identified structures or simulated 3D models of chromosomes and/or molecules.

For image data it is best leaving this to the file system and the native compression of the relevant file formats. At the moment the generation of images is the final step within the pipelines so that they can be used for validation or representation of the results. Allowing the DM API to handle the management of images and where they are located and the serving of the relevant information is currently the best solution. This will be monitored for future changes in technology and whether changes need to happen dependent on the volume of images that are loaded into the VRE. There are options that are available outside of creating archives of sets of related files, including H5IM (https://support.hdfgroup.org/HDF5/doc/HL/RM_H5IM.html) files.

5.1 Sequence mapped data

Many of the pipelines produce BED, Wig, BAM, GFF/GFF3 or TSV files that represent features that are mapped to a particular assembly. Hi-C pipelines also produce aligned data, but this is in the form of an adjacency matrix. When visualising the data often the user will be focusing in on a particular region of a single chromosome at a time. Returning a whole file would therefore be too slow to return to a client and computationally prohibitive to process. It is essential to be able to request genomic regions and transfer the pertinent subsection of data to the client. If more data is required, additional requests can be made.

5.1.1 Sequence Features

Several methods were investigated for the indexing and retrieval of features for given regions. We considered MySQL B-Trees, MySQL R-Trees, Cassandra (http://cassandra.apache.org/), ElasticSearch (https://www.elastic.co/), HDF5 and BigBed/BigWig/Tabix files. This covers a range of SQL and NoSQL options to try and provide the optimal balance between loading, search and retrieval.

Although Cassandra was a potential candidate it was limited by only being able to have a range filter on a single column within the index and so was removed from further testing. For the rest of the storage options testing was performed with a set of ~20,000,000 features derived from the alignment of the dataset DRR000386 (http://www.ebi.ac.uk/ena/data/view/DRR000386) loaded 10 times to represent multiple files (200M features). The FASTQ was aligned to the GCA_000001635.2 (GRCm38/mm10) mouse assembly (http://www.ebi.ac.uk/ena/data/view/GCA_000001635.2) using BWA [Li2009, Li2010]. Each of the FASTQ sequences are ~35bp in length. Table 5.1 shows the results for trying to identify if there are features in a given region of dataset. Each requested region was

1Mb in size and was randomly generated to eliminate the effects of caching within the storage engine or within the file system itself.

Out of the indexes analysed B-Tree, ElasticSearch and Indexed files can directly return the data in the format of the input file (bed in this case). MySQL R-Trees and HDF5 are specially designed for identifying which files have features in a given region. The reason for this secondary index is that programs like BigBed, BigWig and Tabix are designed for the specific purpose of extracting tab separated data from compressed files very efficiently. The limitation from a web perspective is that if there are hundreds of files then many requests are required to be made over the internet to identify if there are features present in a given file. Reducing these calls down to the bare minimum is essential for a responsive application. We have developed a system that returns a list of files with known data within a region allowing a client to request data from that relevant subset of files. Based on the query times this shows that the optimal setup would be to use the HDF5 index and then iterate through the indexed files with BigBed to extract the required data. When using BigBed to retrieve the region from the 10 files it takes 0.1sec. This shows that a mixture of HDF5 to identify relevant files and indexed files are ideally set to act as a backend for a responsive RESTful interface.

Index	Description	Rows	Index size	Query Time
B-Tree	Contains all information from the bed file	206M	30GB	124sec
R-Tree	This is a geometric index on the position	190M	30GB	0.35sec
ElasticSearch	Range query returning JSON matching the bed file	133M	20GB	0.5sec
HDF5	Identifies which files have features within a given range on a base by base case	na	2GB	0.047sec
Indexed File	BigBed file	20M x 10	9.3GB	0.05sec

Table 5.1: *Indexes compared and the data that they stored. Each request was for a 1Mb region and queries were performed 1000 times. The values represent mean execution times for each technology.*

One of the issues to also consider when indexing data is that there is the size of the original data file and the the size of the index. Ideally you do not want the index to be prohibitively large. The size of the uncompressed initial bed file is 1.1GB, resulting in 11GB for 10 duplicate files as part of the test loading. In the case of MySQL (B and R trees) and Elastic Search the sizes of the index were at least twice the size of the original test dataset. This would mean that there would need to be 3 times the storage requirement for each file to accommodate the file and matching index. With the indexed files and HDF5 location file there is only 2 times the storage required as the indexes are roughly the same size as the original data.

Based on the results in Table 5.1 for each of the indexing methods the mg-process-files pipelines (https://github.com/Multiscale-Genomics/mg-process-files) have been developed. These pipelines provide tools to index BED, WIG and GFF3 files with BigBed, BigWig and Tabix respectively and then use HDF5 files to index which files have features for a given region. There have also been readers added to the DM API for reading BigBed, BigWig, Tabix and HDF5 position index files. Interaction with the BigBed and BigWig files uses the pyBigWig module (https://github.com/dpryan79/pyBigWig) to read the files. To create the BigBed and BigWig files we use the bedToBigBed and wigToBigWig executables from UCSC (http://hgdownload.soe.ucsc.edu/admin/exe/). Loading and querying HDF5 files uses the h5py module (http://www.h5py.org/) and GFF3 indexing with Tabix is done using the pysam module (https://github.com/pysam-developers/pysam).

5.1.2 Adjacency Matrixes

These are a special case of range request data which is a value between 2 positions on a chromosome, therefore it is not possible to model within the same format as sequence feature. It also has the additional complexity of the results needing to be calculated for different resolutions and that interactions are sparsely distributed. This is ideally suited to HDF5 files that are designed for handling large, sparse, multidimensional arrays of data and returning only slices of information. This has similar characteristics to the requesting of data from the HDF5 files in 5.1.1. There is the added advantage that these files are compressed and can be easily duplicated if there needs to be horizontal scaling of the RESTful interfaces.

5.2 3D Structures

5.2.1 3D Models

TADbit is able to generate 3D models of fragments of the chromosome [Serra2016]. TADbit outputs multiple models per fragment, where the number of fragments is dependent on the resolution level (1kbp, 10kbp, 100kbp, etc). For each set of models there is a JSON file describing the coordinates for the fragment, a hierarchical clustering of the models including information about which models are the most representative. There is also the matching section of the adjacency matrix embedded within the JSON document. This 3D coordinate data, plus the matching metadata, can get easily stored within a single HDF5 file. This has the advantage that the data can be directly accessed via a python API without requiring all of the JSON files to be generated on the fly, or pre-generated and stored in a zipped archive that would require unzipping.

A pipeline has been written to convert the fully generated set of models into a single HDF5 file. The pipeline is part of the mg-process-files repository, there is also a matching HDF5 file reader included in the DM API.

Smaller nucleic acids and chromatin structures can also be modeled from a sequence in the VRE. In this case, the generated file is stored in PDB format, for 2 main reasons: 1) because it is a relatively small file (single structure), and 2) for compatibility with different tools (visualizers and analysis tools).

5.2.2 Trajectories

Molecular Dynamics trajectories generated in the VRE are stored in different file formats (see Annex section 8.2). When available, binary files (xtc, netcdf) are prefered, because of the reduced file size. However, for compatibility with some of the analysis tools, ASCII-formatted trajectory files (mdcrd, PDB) are also being accepted.

Nucleic acids MD trajectories stored in the BigNASim database will be available through the DM API, exploiting the direct access possibilities offered by the NoSQL platform (see section 3 and reference [Hospital2016]). This will allow the direct retrieval of trajectories and slices of trajectories (both in time and in molecule parts) to the VRE user workspace. The feasibility of using this non-relational approach with MD or CG trajectories computed in the VRE, with a post process by a pipeline following the steps of the work presented for the sequence features (section 5.1.1) will be studied.

5.3 Project Storage Within The VRE

MuG VRE is the web-based interface through which the user interacts with MuG data. MuG VRE is a data-centric interface. It should take information from files, and generate the appropriate environment to visualize or to include them in further analysis. Users can load or create their own data, import it from archives (PDB [Berman2000], BigNASim [Hospital2016], ArrayExpress, ENA), or generate new files resulting from tool and pipeline executions. The DM API is used to track the evolution of these files, as well as to find and update their metadata. This metadata helps the VRE to define the particulars of each file, for instance, to annotate the pipeline from which a file comes from, the set the reference genome assembly of a sequence file, or the establish dependent relationships between a BAM file and its index file (BAI). VRE efficiently queries DM files and display their attributes to the end user, but it also takes advantage of the provenance and the semantic definition of the data to suggest suitable tools, pipelines and visualizers given a file or set of files.

Files in the VRE are organized in projects, groups of files generated by a particular tool or pipeline execution. In the DM model, projects are identified as a particular case of file whose metadata includes the list of all the files that belong to that project, as an additional attribute. In this way , a layer of hierarchy is added for rapidly identifying those interconnected files that, for instance, can be reused all together to launch a new tool execution. These projects are visualized by VRE users as file folders, and in fact, in the file system, the project level corresponds to the working directory where the tool is being executed.

Physically, VRE files are stored in a local repository, a shared disk accessible both, from the VRE instance of the infrastructure, and from the cloud where tool virtual machines are deployed. Yet, the access to such files can also be remote via a RESTful interface which allows the file manipulation from other MuG infrastructures, setting the basis for a true distributed working environment.

5.4 Archives

Within the VRE there is a working level of persistence for files, for long term storage of results then the files should be loaded to the relevant archival services. In the case of sequence reads these should be loaded into the European Nucleotide Archive (http://www.ebi.ac.uk/ena - ENA) [Cochrane2015] where they can be accessioned, stored and distributed. For image data there is the new BioImage service where imaging data, irrespective of source, can be stored and viewed. For nucleic acids MD data, there is the BigNASim database [Hospital2016], not only storing coordinates information but also automatically computing and making available a set of flexibility analysis from the trajectory. Each of these services are able to operate in a private mode so that the data can get loaded and used within the VRE, but can then be easily made publically available upon the point of publication. The BioStudies service can then be used to group together disparate sets of data that form part of the same study with links to ENA, BioSamples [Faulconbridge2014], PDB [Berman2000], etc.

6 CONCLUSIONS

The data management APIs and new pipelines have been designed to be deployed within the COMPSs environment so that they can be easily integrated within the VRE. The pipelines, as with those developed for D4.3, have been designed with the principles of D6.1 in mind. As a result it is possible to efficiently track the progression of a file through workflows within the VRE. On top of the data management, the DM API provides methods for searching the data stored to identify relevant files by various parameters (eg: file type) but also receive a full history of a file and identify the progression from the primary data to the current file.

Where there are files that are more likely to get dynamically requested, involving on the fly slicing of the data, there have been pipelines put in place to implement standard technologies to allow for the quick retrieval of information. The purpose of these sequence feature mapped index files is to reduce the number of requests when there are a large number of potential files. The indexing of the results files has been targeted in a way to optimise the major modes of access that a user will take for retrieving information. In the case of the 3D models these indexes are not only based on returning the relevant models from a given region, but also identifying which of the models are the most representative of the set of potential models.

7 REFERENCES

[Berman2000] Berman, H.M., Westbrook, J., Feng, Z., Gilliland, G., Bhat, T.N., Weissig, H., Shindyalov, I.N., and Bourne, P.E. (2000). The Protein Data Bank. Nucl. Acids Res. *28*, 235–242. https://doi.org/10.1093/nar/28.1.235

[Cochrane2015] Cochrane, G., Karsch-Mizrachi, I., Takagi, T., & Sequence Database Collaboration, I. N. (2015). The International Nucleotide Sequence Database Collaboration. Nucleic Acids Research, gkv1323. https://doi.org/10.1093/nar/gkv1323

[Faulconbridge2014] Faulconbridge, A., Burdett, T., Brandizi, M., Gostev, M., Pereira, R., Vasant, D., Sarkans U, and Parkinson H (2014). Updates to BioSamples database at European Bioinformatics Institute. Nucleic Acids Res 42, D50-2. https://doi.org/10.1093/nar/gkt1081

[Guttman1984] Guttman A. (1984). R-Trees: A Dynamic Index Structure For Spatial Searching. Proceedings of the 1984 ACM, SIGMOD84, 47-57. http://doi.acm.org/10.1145/602259.602266

[Hospital2016] Hospital, A., Andrio, P., Cugnasco, C., Codo, L., Becerra, Y., Dans, P.D., Battistini, F., Torres, J., Goñi, R., Orozco, M. & Gelpí J.L. (2016). BIGNASim: a NoSQL database structure and analysis portal for nucleic acids simulation data. Nucleic Acids Res 44, D272–D278. http://doi.org/10.1093/nar/gkv1301

[Kent2010] Kent, W.J., Zweig, A.S., Barber, G., Hinrichs, A.S., and Karolchik, D. (2010). BigWig and BigBed: enabling browsing of large distributed datasets. Bioinformatics 26, 2204–2207. http://doi.org/10.1093/bioinformatics/btq351

[Langmead2012] Langmead, B., & Salzberg, S. L. (2012). Fast gapped-read alignment with Bowtie 2. Nature Methods, 9(4), 357–359. https://doi.org/10.1038/nmeth.1923

[Leinonen2011] Leinonen, R., Akhtar, R., Birney, E., Bower, L., Cerdeno-Tárraga, A., Cheng, Y., *et al* (2011). The European Nucleotide Archive. Nucleic Acids Research, 39(suppl 1), D28–D31. https://doi.org/10.1093/nar/gkq967

[Li2009] Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., *et al* (2009). The Sequence Alignment/Map format and SAMtools. Bioinformatics, 25(16), 2078–2079. https://doi.org/10.1093/bioinformatics/btp352

[Li2010] Li, H., & Durbin, R. (2010). Fast and accurate long-read alignment with Burrows–Wheeler transform. Bioinformatics, 26(5), 589–595. https://doi.org/10.1093/bioinformatics/btp698

[Li2011] Li H (2011). Tabix: fast retrieval of sequence features from generic TAB-delimited files. Bioinformatics 27, 718–719.

[Marco-Sola2012] Marco-Sola, S., Sammeth, M., Guigó, R., & Ribeca, P. (2012). The GEM mapper: fast, accurate and versatile alignment by filtration. Nature Methods, 9(12), 1185–1188. https://doi.org/10.1038/nmeth.2221

[Serra2016] Serra, F., Baù, D., Filion, G., & Marti-Renom, M. A. (2016). Structural features of the fly chromatin colors revealed by automatic three-dimensional modeling. bioRxiv, 36764. https://doi.org/10.1101/036764

[vanSteensel2010] van Steensel, B., & Dekker, J. (2010). Genomics tools for unraveling chromosome architecture., Genomics tools for the unraveling of chromosome architecture. Nature Biotechnology, Nature Biotechnology, 28, 28(10, 10), 1089, 1089–1095. https://doi.org/10.1038/nbt.1680

8 ANNEXES

8.1 Abbreviations

DAC: Data access committee

EGA: European Genome-phenome Archive

ENA: European Nucleotide Archive

FISH: Fluorescence *in-situ* hybridisation

MINSEQE: Minimum Information about a high-throughput Sequencing Experiment

OME: Open Microscopy Environment

PDB: Protein Data Bank

SRA: Sequence read archive

VRE: Virtual Research Environment

WGBS: Whole Genome Bisulphate Sequencing

8.2 File Formats

8.2.1 FASTA

https://en.wikipedia.org/wiki/FASTA_format

File format for storing sequences (nucleic or peptide) The file consists of a series of entries with a header line beginning with a '>' followed by a stable identifier and any other header parameters. This is followed on the line(s) below with the sequence. There can be multiple sequences in a single file

8.2.2 FASTQ

https://en.wikipedia.org/wiki/FASTQ_format

File format for storing sequences reads (predominantly from high throughput sequencing methods) along with the quality score for each base call. The file consists of 4 lines containing the identifier, the sequence, a description and the quality encoded as a full range of ASCII characters. Derives from the earlier FASTA format.

8.2.3 SAM / BAM

https://samtools.github.io/hts-specs/SAMv1.pdf

SAM is a text based file format for representing a sequence aligned to a reference assembly genome. BAM is a binary compressed and indexed version of a SAM file. Format support is provided by the samtools group who work as part of the Data Working Group of the Global Alliance for Genomics and Health.

8.2.4 GFF3

https://github.com/The-Sequence-Ontology/Specifications/blob/master/gff3.md

File format for describing generic features that exist on a given genetic sequence. These features can consist of a minimum of a single base, but can also be on the forward or reverse strand. The features can also have additional attributes and a matching score.

8.2.5 BED / BigBed

https://genome.ucsc.edu/FAQ/FAQformat#format1

The Browser Extensible Data (BED) format contains the genomic locations of features within the genome. BED is a format where data points are separated by whitespace in text files. The format defines three predictable columns namely chromosome, start and end allowing it to represent most genomic annotations. BED variants may choose to expand the number of columns available to help it represent more complex features such as transcription models. A binary indexed version of this format is available called BigBed providing fast access to genomic location based queries. Format support is primarily provided by the UCSC genome browser group.

8.2.6 WIG (Wiggle)

https://genome.ucsc.edu/goldenpath/help/wiggle.html

Text based file format for associating a numeric value to a given region of position on a reference genome. Wiggle files appear in two formats; fixedStep (for regular data intervals) and variableStep (for irregular data intervals). A binary indexed version of this format is available called BigWig. Format support is primarily provided by the UCSC genome browser group.

8.2.7 CGMap and ATCGMap

https://github.com/BSSeeker/BSseeker2/blob/master/README.md

This is a file format for disseminating methylome data [Guo2013]. Both files includes the CpG and CpH sites. The ATCGMap also includes the positions of all features for all bases on both strands.

8.2.8 HDF5

https://support.hdfgroup.org/HDF5/

A file format designed to handle large data arrays. The file can handle the multiple sparse arrays easily in a very compressed format. Manipulation of records stored in this format must be handled by dedicated libraries.

8.2.9 JSON

http://www.json.org/

File format used to handle attribute-value pairs in a semi-human readable way. Derives from JavaScript, but is openly accessible by a large number of programming languages. Due to its flexibility in representing any data schema and ubiquitous language support JSON is used as a fast way to deliver data to a client browser (Chrome/Firefox/Edge) or to server processes over RESTful APIs.

8.2.10 PDB

http://www.wwpdb.org/documentation/file-format

Plain text file format for the representation of 3D molecular structures. The format has reached a stable point and was frozen in November 2012.

8.2.11 xtc / NetCDF / mdcrd

http://ambermd.org/formats.html#trajectory

http://manual.gromacs.org/online/xtc.html

Molecular Dynamics (MD) and Coarse-Grained (CG) trajectory files, containing cartesian 3D coordinates (x,y,z) information for all the elements (usually atoms) of a molecule for all the snapshots in a particular time slice. NetCDF (binary) and mdcrd (ASCII) are specific for AMBER MD package, whereas xtc (binary) is specific for GROMACS MD package.

8.2.12 top / tpr / parmtop

http://ambermd.org/formats.html#topology

http://manual.gromacs.org/online/top.html

http://manual.gromacs.org/online/tpr.html

Molecular Dynamics (MD) and Coarse-Grained (CG) topology files, containing information about the molecule such as number of elements (usually atoms), type, name, bonds, angles, dihedrals, etc. They are indispensable to analyse and visualise trajectories stored in the formats presented in the previous section. Top (ASCII) and tpr (binary) are specific for GROMACS MD package, whereas parmtop is specific for AMBER MD package.

8.2.13 cpt / rst

http://ambermd.org/formats.html#restart

http://manual.gromacs.org/online/cpt.html

GROMACS (cpt) and AMBER (rst) checkpoint and restart files used in molecular modelling simulations. They allow to extend a trajectory from the last snapshot computed, and also to restart a simulation from the last snapshot computed after a crash.

8.2.14 GEM

http://algorithms.cnag.cat/wiki/The_GEM_library

GEM is a binary index file format for use indexing assemblies to aid in the alignment of FASTQ data [Marco-Sola2012].

8.2.15 Bowtie2 Index

http://bowtie-bio.sourceforge.net/bowtie2

This consists of 6 files type (.1.bt2, .2.bt2, .3.bt2, .4.bt2, .rev.1.bt2, .rev.2.bt2) that are used for the alignment of high throughput sequencing reads to a given genome assembly.

8.2.16 BWA Index

http://bio-bwa.sourceforge.net/

This actually covers 5 file types that describe the index (amb, ann, bwt, pac, sa). These are used for the alignment of high throughput sequence reads to a given genome assembly