Processing of big heterogeneous genomic datasets for tertiary analysis of Next Generation Sequencing data

Marco Masseroli[1,*], Arif Canakoglu[1], Pietro Pinoli[1], Abdulrahman Kaitoua[2], Andrea Gulino[1], Olha Horlova[1], Luca Nanni[1], Anna Bernasconi[1], Stefano Perna[1], Eirini Stamoulakatou[1], Stefano Ceri[1].

[1] Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Piazza Leonardo da Vinci 32, 20133, Milan, Italy.

[2] The German Research Center for Artificial Intelligence (DFKI), Alt-Moabit 91c, 10559 Berlin, Germany.

Associate Editor: XXXXXXX

Received on XXXXX; revised on XXXXX; accepted on XXXXX

Abstract

Motivation: We previously proposed a paradigm shift in genomic data management, based on the Genomic Data Model (GDM) for mediating existing data formats and on the GenoMetric Query Language (GMQL) for supporting, at a high level of abstraction, data extraction and the most common data-driven computations required by tertiary data analysis of Next Generation Sequencing datasets. Here, we present a new GMQL-based system with enhanced accessibility, portability, scalability and performance.

Results: The new system has a well-designed modular architecture featuring: i) an intermediate representation supporting many different implementations (including Spark, Flink, and SciDB); ii) a high-level technology-independent repository abstraction, supporting different repository technologies (e.g., local file system, Hadoop File System, database, or others); iii) several system interfaces, including a user-friendly Web-based interface, a Web Service interface, and a programmatic interface for Python language. Biological use case examples, using public ENCODE, Roadmap Epigenomics and TCGA datasets, demonstrate the relevance of our work.

Availability: The GMQL system is freely available for non-commercial use as open source project at: http://www.bioinformatics.deib.polimi.it/GMQLsystem/

Contact: marco.masseroli@polimi.it

1 Introduction

Next Generation Sequencing (NGS) allows the production of multiple high-throughput datasets regarding, among others, genome sequencing (DNA-seq), transcriptome profiling (RNA-seq), DNA-protein interaction assessment (ChIP-seq) and epigenome characterization (ChIP-seq, BS-seq, DNase-seq and FAIRE-seq) (Goodwin *et al*., 2016). The size of datasets and the complexity of computations motivate the use of parallel and distributed computing to achieve scalability and performance (Eric *et al*., 2010; Schmidt and Hildebrandt, 2017). Multiple international sequencing projects are producing (epi)genomic feature data, extracted through standardized primary and secondary analysis pipelines (i.e., genome alignment and feature calling of NGS raw data) and available at public repositories (Del Chierico *et al*., 2015); among them:

- The *Encyclopedia of DNA elements* (ENCODE) (ENCODE Project Consortium, 2012).
- *The Cancer Genome Atlas* (TCGA) (Cancer Genome Atlas Research Network *et al*., 2013).
- The 100,000 Genomes Project (Siva, 2015).
- Roadmap Epigenomics (Bernstein *et al*., 2010)

Availability of advanced NGS technologies and of open repositories of primary and secondary data brings about a new need of tools for NGS tertiary analysis, i.e., multi-sample integration of genome-wide, heterogeneous features and known annotations. Tertiary data analysis is still mostly performed through ad hoc scripts, typically invoking multiple software tools for specific operations; these tools are not usually designed for large amounts of data and require converting the outputs of one tool into the inputs of another (an overview of existing methods and systems is in the next Section *Comparison with other systems*). Thus, there is an increasing need of powerful, user-friendly and versatile environments for tertiary data analysis, that can enable scientists and bioinformaticians to focus on the biological questions and on the design of computational experiments, rather than on how to implement their computational steps across multiple systems and data formats.

For addressing these challenges, we introduced GenoMetric Query Language (GMQL) (Masseroli *et al*., 2015), a high-level declarative language allowing the expression of queries over genomic regions and their metadata, in a way similar to what can be done with the well-known Relational Algebra and Structured Query Language (SQL) over a relational database. GMQL uses the Genomic Data Model (GDM) (Masseroli *et al*., 2016) which is based on the notion of *genomic region*, mediates existing data formats, and covers also *metadata* of arbitrary structure; thanks to these features, GDM is capable to support data interoperability, by describing semantically heterogeneous data. As a proof of concept to demonstrate GDM and GMQL features, we originally implemented the translation of GMQL queries to Pig Latin (Olston *et al*., 2008), a high-level data-flow language for batch processing of large data sets; we used *Apache Pig* (http://pig.apache.org/), an open source platform for analyzing large data sets, to manage and execute Pig Latin scripts. However, the link between GMQL and the implementation language was too tight, and Pig Latin limited the potential of GMQL, with unsatisfactory results for the increasing requirements of genomic big data evaluations.

Here, we illustrate and discuss a novel GMQL system that meets efficiency, flexibility, and usability requirements. The new system supports queries (i.e., scripts) comparing billions of genomic regions, mainly on the basis

[*] To whom correspondence should be addressed.

of metric properties but also of arbitrary region attributes and of metadata content. Besides extending and enhancing GMQL operators and functionalities, we abstracted GMQL from the implementation layer and developed a novel system with a language independent modular architecture: a GMQL compiler generates Directed Acyclic Graphs (DAGs) as intermediate representation of GMQL scripts, where DAG nodes are implemented using cloud computing technologies. With this approach, we support multiple usage scenarios for GMQL, including Web based interfaces or language embeddings, and multiple implementations using different cloud computing engines.

Additionally, we defined and implemented new data binning strategies (described in details elsewhere (Kaitoua *et al*., 2017; Cattani *et al*., 2017a)) which enhance scalability and parallelism, by adapting genomic data processing to the nature of the cloud architecture. This allows GMQL executions either on a cloud environment, or on a single Java virtual machine (JVM) (i.e., local execution mode); the latter one avoids the complexity of allocating resources and running on a cluster, but it is adequate only for limited data size and not suitable for big data processing. For easy access to the new GMQL system, we developed REST Web Services and a comprehensive Web interface; they allow easy use of the system and versatile processing of different genomic features to extract candidate targets for knowledge discovery. The new GMQL system is available as open source project in the GitHub platform and can be tested through the REST and Web interfaces (see: http://www.bioinformatics.deib.polimi.it/GMQLsystem/, where also examples of use and full documentation are available).

2 The new GMQL and its processing

GMQL relevantly evolved and enhanced since its previous version (Masseroli *et al*., 2015). All its operators were redesigned, with a unified syntax, and now all of them operate both on regions and on metadata; originally they were specialized to work either on regions or on metadata, with the exception of the JOIN, DIFFERENCE and MAP operators. Previously, the language did not support attribute projection (as the PROJECT operator was used for selecting regions) and grouping (groups were implicitly created by the AGGREGATE operator); now, PROJECT, GROUP and EXTEND operators provide orthogonal functionalities. SUMMIT, originally an independent operator, is now an extension of the COVER operator, which also includes FLAT as another extension; for what concerns the JOIN operator, now genometric predicates are more orthogonal and the possibility of joining arbitrary region attributes was added. For a full account of the differences, compare the documentations of the two GMQL releases (the earlier release is still documented at: http://www.bioinformatics.deib.polimi.it/GMQL/).

In the new GMQL system, the translation of a GMQL query to an execution plan is achieved through a two-step procedure. First, the GMQL query is compiled to an intermediate representation (IR); next, the IR is interpreted by the GMQL execution engine and an actual execution plan is produced and run. To clearly present these elements, throughout this Section we discuss the following exemplary GMQL query (videos showing a step-by-step execution of this query, using either the GMQL public Web interface or Python library next described, are available at: http://www.bioinformatics.deib.polimi.it/geco/?video):

```
myExperiment = SELECT() UPLOADED;
myData = COVER(2, ANY) myExperiment;
genes = SELECT(annotation_type == "gene" AND
        provider == "RefSeq") HG19_BED_ANNOTATIONS;
onGenes = JOIN(distance < 0; output: right)
        genes myData;
mutations = SELECT(type == "SNP") ICGC_REPOSITORY;
geneMutationCount = MAP() onGenes mutations;
MATERIALIZE geneMutationCount INTO result;
```

The query uses COVER, JOIN and MAP, the most interesting domain-specific operations of GMQL (Masseroli *et al*., 2015). Given three replicate samples of a ChIP-seq experiment (selected by the user), the specified COVER operation extracts high-confidence regions (confirmed in at least two of the input samples), the JOIN operation identifies which of these regions overlap with genes in the NCBI Reference Sequence Database (RefSeq) (https://www.ncbi.nlm.nih.gov/refseq/), and the MAP operation counts the SNP mutations (from the International Cancer Genome Consortium – ICGC (http://icgc.org/)) present in each of such regions. The query combines experimental data uploaded by the user, gene annotations for human genome assembly hg19 from RefSeq, and public mutation data from the ICGC repository; the datasets are described using the GDM integrative data model (Masseroli *et al*., 2016).

2.1 GMQL compiler

The compiler reads a GMQL query, validates its syntax, and – if the query is syntactically correct – builds its syntactic tree and performs its semantic validation. In this latter phase, the compiler checks that all GMQL variables (i.e., datasets) defined in the query are manipulated in coherent and consistent way (e.g., no undefined variable is referenced, only those features that are present in the region dataset are mentioned by the query). If any of such tests fails, the compiler notifies the user with an error message reporting the type and position in the query of the encountered problem; otherwise, it produces the intermediate representation of the query. The compiler is built by leveraging on functional features of the Scala programming language (https://www.scala-lang.org/).

2.2 Intermediate representation

The IR is an encoding of a GMQL query that omits all its syntactic details and does not encapsulate any of the execution engine peculiarities. Thus, the IR acts as an interface between the GMQL language and the engine; thanks to such abstraction, we can reuse the very same engine to execute any GMQL dialect (aside of GMQL we developed Scala, Python and R/Bioconductor application programming interfaces (APIs). Conversely, the IR allows implementing alternative execution engines; after testing alternative implementations based on Apache Spark (https://spark.apache.org/), Apache Flink (https://flink.apache.org/) and SciDB (http://scidb.sourceforge.net/), our preferred one uses Apache Spark. Such vast set of combinations of GMQL interfaces and execution engines would not have been possible without the IR. Figure 1 shows the IR of our exemplary GMQL query.

Technically, an IR is a directed acyclic graph, where nodes represent elementary parametric operations (e.g., a filter on region data or a filter on metadata) and edges represent the flow of the execution. Nodes can be clustered in two categories: nodes manipulating metadata (in red in Figure 1) and nodes manipulating region data (in blue in Figure 1). Their synchronization is granted by a wise graph construction.

The use of an intermediate representation gives several optimization opportunities: semantically equivalent permutations of the operators can be automatically discovered, different implementations of the same operation can be used depending on input or system characteristics, and advanced optimizations that rely on the differences between nodes can be applied. The latter one is the case of the powerful *meta-first execution strategy,* in which all metadata nodes are evaluated before all region nodes to determine the only samples that contribute to the result due to all the metadata predicates computed by the query; such information is provided to the data loaders for selective data access, and in many cases it drastically reduces the amount of (big size) region samples loaded to the execution engine and then processed, with corresponding high decrease of execution time. This run-time optimization is not supported by any other genomic data management system.

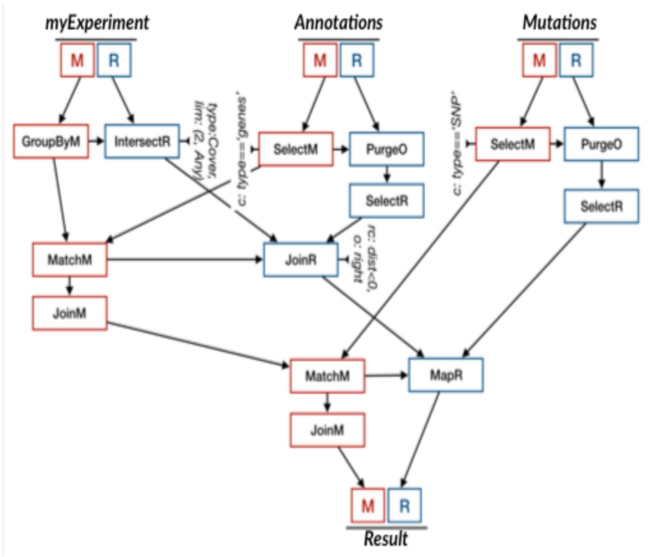


Fig. 1: Intermediate representation of the exemplary GMQL query; metadata nodes are in red, region data nodes in blue.

3 GMQL system architecture

The GMQL system is organized according to a four-layer architecture:

i) The *Access layer* supports a user-friendly *Web interface*, a *shell command line interface*, and several *Web Services* / *APIs* for supporting access to GMQL system resources. The Scala API (https://github.com/DEIB-GECO/GMQL/wiki/GMQL-APIs) provides the main functionalities for connecting to the GMQL system, as all the other APIs are built from it.

ii) The *Engine layer* includes the *Compiler*, the *DAG Manager* (for supporting the creation and dispatch of DAG operations to other system components), the *Server Manager* (for multi-user execution over heterogeneous implementations and environments, such as local vs. distributed), the *Repository Manager* (for accessing a data repository on heterogeneous file systems), and the *Launcher Manager* (for launching the executions for different implementations; currently, we have three launchers: Local Launcher, Cluster Launcher, and SciDB Launcher.)

iii) The *DAG Implementation layer* includes the implementations (abstract classes) of the DAG operations for the Spark, Flink and SciDB engines. The *Spark Implementation* is the default one.

iv) The *Repository Implementation layer* includes the *Local File System* (LFS) repository, used on a single local machine architecture; the *Hadoop Distributed File System* (HDFS), used on a server-based architecture; and the *Remote File System* (RFS), used in a cluster-based architecture.

An implementation of the GMQL system open for public use at http://www.gmql.eu/ is installed on a cluster at CINECA (a not-for-profit Consortium including 70 Italian Universities, 6 Research Consortia, and the Ministry of Education, University and Research of Italy (MIUR)). We opted for a deployment strategy based on an application server and a cluster of machines for execution over Spark engine and HDFS. The application server receives a GMQL script from the Web Service / Web interface and compiles the script producing a DAG with the intermediate representation of the operations. The DAG is then serialized and sent from the application server to the cluster for processing; the cluster consists of three nodes, each using 40 virtual CPUs, 125 GB of RAM and 3 TB of disk. Thus, the distributed computing environment consists of a total of 120 cores, 375 GB of RAM and 9 TB of disk.

3.1 Web interface

The GMQL Web interface has been designed with the goal of providing a user-friendly intuitive environment for bioinformaticians and biologists. It supports multiple functionalities, including: management and browsing of public and private datasets uploaded in the associated repository, building GMQL queries upon them, visualizing the GMQL processing output by direct connection to the UCSC Genome Browser (Karolchik *et al*., 2012) or to integrated heat map viewers, and downloading the data locally for further data analysis and use of client-side applications such as offline genome browsers (e.g., IGB (Freese *et al*., 2016), or IGV (Robinson *et al*., 2011)).

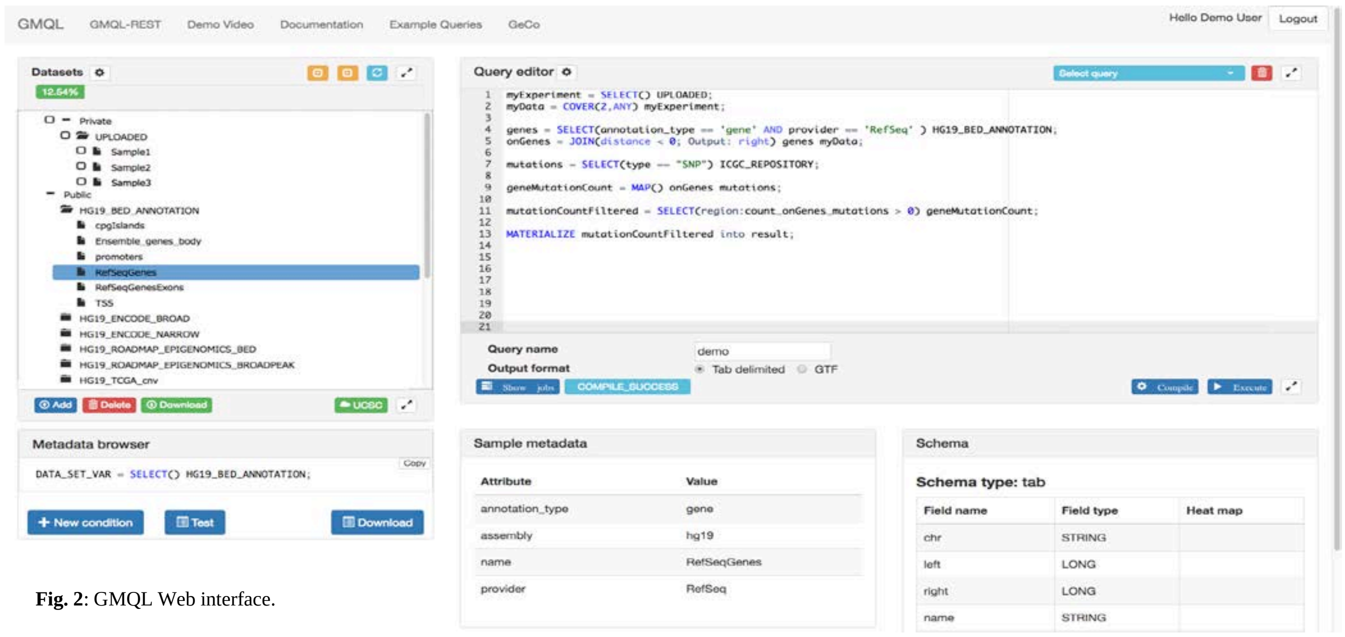
The Web interface uses the Web Service interface, a REST API (specified using XML or JSON) which also supports libraries for integrating GMQL with the programming environments of Python and R/Bioconductor, and for supporting interactions from Galaxy workflows. In this way, GMQL queries are integrated within the most typical contexts of use in bioinformatics. The Web interface supports both registered users (authenticated through a password) and guest users who are provided with limited resources. After login, the user is presented with an interface described in Figure 2. The screen is divided into different sections to support different functionalities.

In the *Datasets* section (upper left corner), users browse private and public datasets. Private datasets are those created by the specific user as results of upload operations or previously executed GMQL queries. Public datasets include predefined, read-only datasets from public repositories – currently we support datasets from ENCODE, Roadmap Epigenomics and TCGA, as well as annotations from UCSC (partially), RefSeq and GENCODE. Our TCGA curation is described in (Cumbo *et al*. 2017); our work on the standardization of a minimal set of well-defined metadata for public repositories is presented in (Bernasconi *et al*., 2017). Users can select samples from both private and public datasets; in Figure 2, the user has selected the RefSeqGenes sample in the HG19_BED_ANNOTATION dataset; note that GDM provides the same representation for both annotation and genomic feature data.

The two *Sample metadata* and *Schema* sections (lower center and right corner) respectively show the metadata attribute-value pairs and the schema (i.e., the attribute order and type) of the genomic regions in the selected sample. We recall that in GDM each sample combines metadata and genomic regions, both for the initial datasets loaded from the repository and for the datasets computed as result of a GMQL query.

Below the Dataset section, the *Metadata browser* section helps browsing metadata and filtering samples of the selected dataset; at the same time, it composes the correspondent SELECT statement based on sample metadata, ready to be used in a GMQL query. Metadata of the selected samples are shown in a pop-up window; Figure 3 shows the case of two samples from the HG19_BED_ANNOTATION dataset.

In the *Query editor* section (upper right corner) the user can compose a GMQL query, compile it and then execute it. User-friendly dialog boxes support the user in detecting and correcting errors in the composed GMQL query, and inform about the progression of a GMQL execution over the used engine and implementation. The user can compile queries without executing them, to check their correctness. When execution is launched, the log viewer of the query execution job becomes visible, showing the execution status; more information can be extracted from each query log.



- Private
  - UPLOADED
    - Sample1
    - Sample2
    - Sample3
- Public
  - HG19_BED_ANNOTATION
    - cpgIslands
    - Ensemble_genes_body
    - promoters
    - RefSeqGenes
    - RefSeqGenesExons
    - TSS
  - HG19_ENCODE_BROAD
  - HG19_ENCODE_NARROW
  - HG19_ROADMAP_EPIGENOMICS_BED
  - HG19_ROADMAP_EPIGENOMICS_BROADPEAK
  - HG19_TCGA_cnv

```
myExperiment = SELECT() UPLOADED;
myData = COVER(2,ANY) myExperiment;

genes = SELECT(annotation_type == 'gene' AND provider == 'RefSeq' ) HG19_BED_ANNOTATION;
onGenes = JOIN(distance < 0; Output: right) genes myData;

mutations = SELECT(type == "SNP") ICGC_REPOSITORY;

geneMutationCount = MAP() onGenes mutations;

mutationCountFiltered = SELECT(region:count_onGenes_mutations > 0) geneMutationCount;

MATERIALIZE mutationCountFiltered into result;
```

Query name: demo
Output format: [✓ Tab delimited] GTF

```
DATA_SET_VAR = SELECT() HG19_BED_ANNOTATION;
```

Attribute	Value
annotation_type	gene
assembly	hg19
name	RefSeqGenes
provider	RefSeq

Schema type: tab

Field name	Field type	Heat map
chr	STRING	
left	LONG	
right	LONG	
name	STRING	

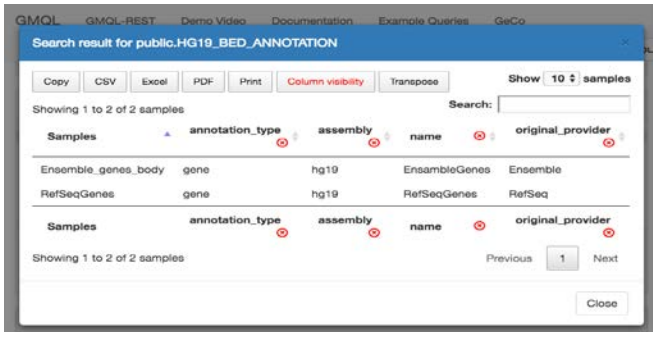
Fig. 2: GMQL Web interface.

3.2 PyGMQL Python library

GMQL supports also PyGMQL, a Python interface which enables the user to perform GMQL queries in an interactive and exploration-driven fashion. PyGMQL is simply installed through the classical package installer system pip, just by typing: *pip install gmql*.

PyGMQL embeds a GMQL API which supports both a local query execution, using the API as local back-end, or a remote execution, by sending the query (as a serialized graph structure) to a remote server where the GMQL engine is installed. An excerpt of the example query, which performs the join operation, is expressed in a Python program as follows, where annotations are taken from a remote server and peak data are loaded from the local disk:

```
import gmql as gl
gl.set_remote_address("http://www.gmql.eu/gmql-rest/")
gl.login()
gl.set_mode("remote")
d_remote = gl.load_from_remote("HG19_BED_ANNOTATION",
    owner="public")
d_genes = d_remote [d_remote ["annotation_type"] == "gene"]
d_local = gl.load_from_path("./path/to/local/narrowpeak_dataset/")
d_join = d_genes.join(d_local, [gl.DL(0)], "right")
result = d_join.materialize()
```



Search result for public.HG19_BED_ANNOTATION

Showing 1 to 2 of 2 samples

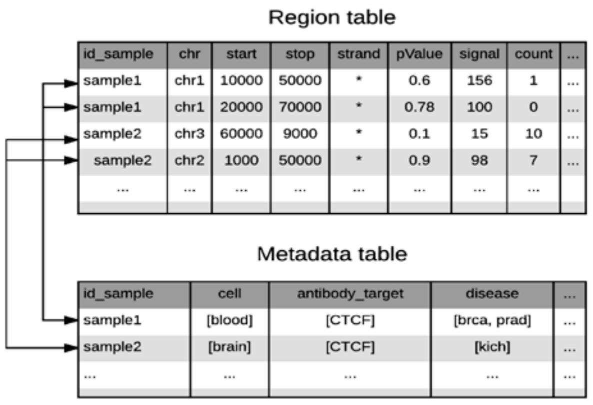
Samples	annotation_type	assembly	name	original_provider
Ensemble_genes_body	gene	hg19	EnsambleGenes	Ensemble
RefSeqGenes	gene	hg19	RefSeqGenes	RefSeq

Showing 1 to 2 of 2 samples

Fig. 3: Metadata browsing for selected samples.

The result dataset of a PyGMQL script (in this case the variable *result*) is automatically loaded in memory as a Python structure called GDataFrame, which extends a DataFrame (i.e., a table which has a row index and a set of columns) of *pandas* (https://pandas.pydata.org/), a well-known data analysis library for Python, thus supporting interoperability with other Python code. A GDataFrame is composed of two *pandas* DataFrames, shown in Figure 4:

- A *region table*, where each row is a genomic region and every column represents a region genomic coordinate or feature; the index of the DataFrame is the list of sample identifiers, which can be repeated;
- A *metadata table*, where each row represents a dataset sample and every column represents a metadata attribute; the index of the DataFrame is the list of sample identifiers, without repetitions.


Region table

id_sample	chr	start	stop	strand	pValue	signal	count	...
sample1	chr1	10000	50000	*	0.6	156	1	...
sample1	chr1	20000	70000	*	0.78	100	0	...
sample2	chr3	60000	9000	*	0.1	15	10	...
sample2	chr2	1000	50000	*	0.9	98	7	...
...	...	...	...	...	...	...	...	

Metadata table

id_sample	cell	antibody_target	disease	...
sample1	[blood]	[CTCF]	[brca, prad]	...
sample2	[brain]	[CTCF]	[kich]	...
...	...	...	...	...

Fig. 4: Visual representation of a GDataFrame. Note that coherency between region and metadata tables is kept by the common *id_sample* column, and that each sample has one row in the metadata table, whose attributes can have multiple values. Note as well that GMQL results carry metadata both of the initial datasets and generated during query processing.

It is also possible to convert an arbitrary pandas DataFrame to a GMQL variable and use it in a GMQL query. Thanks to its underlying pandas implementation, the GDataFrame can be used as a starting point for complex data manipulation using specific Python libraries for data analysis (like NumPy (http://www.numpy.org/) or SciPy (https://www.scipy.org/)), or for machine learning / deep learning (like scikit-learn (http://scikit-learn.org/), TensorFlow (https://www.tensorflow.org/) or Keras (https://keras.io/)). The user is encouraged to use the PyGMQL library with the support of a Jupyter Notebook (http://jupyter.org/), a programming environment for data exploration; its usage improves reproducibility of results, their sharing and visualization.

4 Example use case

In this section, we show a typical GMQL query over multiple heterogeneous genomic features; more examples of biological interest are in the Documentation section at http://www.bioinformatics.deib.polimi.it/GMQLsystem/ and in the Supplementary material, where GMQL is compared to STQL (Zhu *et al*., 2017). In the following use case, genomic features from TCGA patient data are combined with their clinical and biospecimen metadata, thanks to their availability in BED format (as provided by TCGA2BED (Cumbo *et al*. 2017)), which is compatible with GDM. The example focuses on Breast Invasive Carcinoma (BRCA) patients (the ones with the highest amount of data in TCGA) and extracts expressed genes, DNA methylations (which generally repress gene expression) and somatic mutations close to methylated expressed genes. Comprehensively considering genomic, epigenomic and transcriptomic data of cancer patients provides a view of the patients' complex biomolecular system, which may lead to interesting findings.

The use case and its GMQL query are formulated as follows: *"In TCGA data of BRCA patients, find the DNA somatic mutations within the first 2000 bp outside of the genes that are both expressed with FPKM > 3 and have at least a methylation in the same patient biospecimen, and extract these mutations of the top 5% patients with the highest number of such mutations."*

```
EXPRESSED_GENE = SELECT(manually_curated__cases__disease_type
    == "Breast Invasive Carcinoma"; region: fpkm > 3.0)
    GRCh38_TCGA_gene_expression;
METHYLATION = SELECT(manually_curated__cases__disease_type ==
    "Breast Invasive Carcinoma") GRCh38_TCGA_methylation;
MUTATION = SELECT(manually_curated__cases__disease_type ==
    "Breast Invasive Carcinoma")
    GRCh38_TCGA_somatic_mutation_masked;

GENE_METHYL = JOIN(distance < 0; output: left_distinct; joinby: bio-
    specimen__bio__bcr_sample_barcode) EXPRESSED_GENE
    METHYLATION;
MUTATION_GENE = JOIN(distance <= 2000, distance >= 0; output:
    left_distinct; joinby: biospecimen__bio__bcr_sample_barcode)
    MUTATION GENE_METHYL;

MUTATION_GENE_count = EXTEND(mutation_count AS COUNT())
    MUTATION_GENE;
MUTATION_GENE_top = ORDER(mutation_count DESC; meta_topp: 5)
    MUTATION_GENE_count;
MATERIALIZE MUTATION_GENE_top INTO MUTATION_GENE_top;
```

The query is divided into three sections. Using the SELECT operator, the first one extracts relevant samples from three TCGA datasets (gene expressions, DNA methylations, somatic mutations); the second one combines the extracted samples and metrically evaluates the localization of their genomic regions, by means of two JOIN operations, to produce the relevant mutations searched; the third one counts them and selects those of the most mutated patients.

Specifically, the first JOIN operator applies on expressed gene and DNA-methylation datasets. It first combines samples based on the equivalence of their metadata *biospecimen__bio__bcr_sample_barcode* attribute (the TCGA biospecimen identifier); then, from every pair of samples of each biospecimen, it extracts the expressed gene regions that overlap at least a methylation site in the paired DNA methylation sample. Similarly, the second JOIN operator applies on the extracted expressed and methylated genes in each sample and on the entire BRCA mutation dataset of TCGA; in each sample of the latter one, it finds the DNA somatic mutations occurring within the first 2,000 bp upstream or downstream of any of the expressed methylated genes extracted in the paired sample of the same biospecimen. Then, the EXTEND operator uses the COUNT() aggregate function to determine the number of these mutations in each sample, the ORDER operator ranks the samples according to such number and extracts the top 5% samples with the highest number of these somatic mutations, and finally the MATERIALIZE operator returns the result. Note that this complex query is simply expressed through a few GMQL statements, also thanks to the GMQL implicit iteration over all the samples even matched through their metadata.

At the time of writing (March 2018), the query was executed over all 11,091 gene expression, 12,218 DNA methylation and 10,188 somatic mutation samples publicly available in TCGA, for a total of 56.5 GB, 1.3 TB and 2.3 GB of data, respectively. The query initially selected 1,222 samples of expressed gene data, 1,234 samples of DNA methylation data, and 985 samples of DNA somatic mutation data of TCGA BRCA patients, containing a total of 11,847,376 expressed gene regions, 358,803,211 methylation sites, and 363,521 DNA mutations, respectively.

The combination of each biospecimen's gene expression and DNA methylation data identified 1,207 breast cancer patient samples presenting methylated expressed genes, with an average of 7,822 of such genes for each identified biospecimen. Thanks to the TCGA patients' clinical data reported in the available sample metadata, which GDM seamlessly manages and GMQL carries on during the processing, these patients can be clinically characterized. In particular, they have an *average age at diagnosis* of 58.27 years; 551 of them received *radiation therapy*, whereas 453 did not, and for 203 of them it is unknown; 820 patients are *estrogen receptor positives* and 236 *negatives*; 712 are *progesterone receptor positives* and 349 *negatives*.

Then, the query extracts 612 biospecimens having somatic mutations occurring within the first 2000 bp outside of the same biospecimen's expressed and methylated genes. Finally, these mutations in each biospecimen are counted (their average number per biospecimen is 2.9), and the mutations of the top 5% patient biospecimens with the highest number of such somatic mutations are selected (their average number per biospecimens is 20.87). Table S2 in the Supplementary material reports an excerpt of the metadata attributes and of their values associated with the selected patients. Notably, the top patient biospecimen has 120 mutations, about three times of the ones of the second top patient, who was first diagnosed with BRCA when was about 30 years younger; all patients but 8 are positives to progesterone and/or estrogen receptor, and 9 of them received radiation therapy whereas 11 did not.

The whole execution of this example query on the public GMQL system installation, where the entire public TCGA datasets are available, lasted only 57 minutes; execution time is low when compared to the big amount of samples and genomic regions processed, and to the complexity of the processing.

The same GMQL query can be directly applied on other types of patients or datasets, just by changing the SELECT operator parameters. Note that the result dataset includes both genomic somatic mutations and clinical metadata of the finally selected patients. The former ones indicate interesting somatic mutations that could be associated with breast cancer (which can be further inspected, e.g., using genome browsers); the latter ones allow tracking the provenance of resulting samples and ease the biomedical interpretation of the results, facilitating also result sample stratification and further evaluations. This association between processed genomic data and their biological/clinical metadata is not supported by other system currently available, and represents one of the relevant aspects of GDM and GMQL. We previously developed an analytic tool that takes specific advantage of such association (Jalili *et al*., 2017).

5 Comparison with other systems

In the *Introduction* Section, we observed that tertiary data analysis is still mostly performed through ad hoc scripts, typically invoking multiple software tools for specific operations. BEDTools (Quinlan and Hall, 2010) and BEDOPS (Neph *et al*., 2012) gained relevance in the bioinformatics community; they efficiently process region data in BED format of individual or paired samples, but require verbose scripts for multiple sample processing, with lower performance. In Supplementary material of (Masseroli *et al*., 2015) we thoroughly compared them with GMQL. Also packages of R/Bioconductor (https://www.bioconductor.org/) have been proposed for tertiary analysis (Huber *et al*., 2015); they facilitate typical specific operations, but require to perform them through scripts and are not suitable for big data processing. Recently, the Genomic Region Operation Kit (GROK) (Ovaska *et al*., 2013) has been proposed; it adopts a genomic region abstraction (able to represent reads, variants, mutations, and other genomic features) and provides a set of region operations, based on a mathematical formalism founded on set algebra, delivered as a library. GORpipe (Guðbjartsson *et al*., 2016) embeds relational operations within scripts, thereby injecting some declarative aspects (such as selects, joins and merges).

In the last years, high performance parallel and cloud computing approaches, mainly founded on Hadoop (Shvachko *et al*., 2010) and *MapReduce* (Dean and Ghemawat, 2010) frameworks, have been adopted also in bioinformatics (O'Driscoll *et al*., 2013; Mrozek *et al*., 2014; Mrozek *et al*., 2016), with a focus on primary and secondary analysis of NGS raw data. Among them, *SparkSeq* (Weiwiorka *et al*., 2014) has been proposed as general purpose tool to process DNA and RNA sequencing data using the *Apache Spark* engine. Notably, the Genome Analysis Toolkit (GATK) (McKenna *et al*., 2010) has gained a leading role for variant and SNP calling secondary analysis; its new version 4 (GATK4) has recently been released as open source on the GitHub platform (https://www.broadinstitute.org/news/broad-institute-release-genome-analysis-toolkit-4-gatk4-open-source-resource-accelerate).

Few informatics systems support tertiary data analysis; among them, DeepBlue (Albrecht *et al*., 2016) provides integrated access to genomic datasets (ENCODE, Roadmap Epigenomics, and datasets produced within the BluePrint Consortium), annotated by a small collection of curated metadata; it also provides some programmatic data retrieval for selecting and aggregating processed datasets. SciDB (Brown, 2010) is a computational multi-dimensional, array-based database engine optimized for fast data selection and aggregation, required by most scientific applications; it recently developed extensions for genomics (https://www.paradigm4.com/).

Some query languages compare more closely to GMQL as they introduce a relational paradigm into genomic computing; all of them use relational operations over regions, none of them applies such operations to metadata – thereby lacking the ability to constructively assign metadata to query results, a distinguishing feature of GMQL. Among them, systems based on the Structured Query Language and its extensions (Kozanitis *et al*., 2014; Zhu *et al*., 2017) have been recently proposed. The Genomic Query Language (GQL), presented in (Kozanitis *et al*., 2014), applies to raw data and includes genomic feature calling; this approach creates reproducibility issues when compared to more conventional pipelines. Other recent systems, including BigDataScript (Cingolani *et al*., 2015) and GESALL (Roy *et al*., 2017), focus on scripting languages and systems for implementing and enhancing data pipelines; in particular, GESALL is a system, adopted at The New York Genome Center (NYGC), capable of wrapping existing systems and autonomously determining the best parallelism, spotting superlinear speed-ups, but also sublinear steps with loss of performance.

The Signal Track Query Language (STQL), lately proposed in (Zhu *et al*., 2017), is instead focused on tertiary analysis and closer in design to GMQL: STQL tracks correspond to GMQL samples. STQL uses the classic Select-From-Where SQL nesting rather than sequences of distinct operations. The main difference is that GMQL queries produce multi-sample results, each loaded to the file system as a separate file and carrying sample metadata that describe both query input and processing, whereas STQL produces a single track. Another difference is that STQL is implemented directly on Apache Hive (https://hive.apache.org/) warehousing (optimized for supporting some STQL primitives) and hence depends on a specific cloud engine, with higher upkeep cost, while our approach maps DAG operations to engines, with an engine-independent execution workflow (we implemented GMQL on Spark, Flink and SciDB). A thorough comparison between STQL and GMQL is presented in the Supplementary material, where we compare STQL queries provided in supplemental material of (Zhu *et al*., 2017) to equivalent GMQL queries, serving as well to describe biological examples of GMQL use.

Noteworthy, FireCloud (https://software.broadinstitute.org/firecloud) is a cloud-based platform developed at the Broad Institute for supporting cancer genome analysis, primarily focused on secondary analysis. FireCloud pipelines are specified in Workflow Specification Language (WSL), a simple workflow language supporting calls to a variety of tools, and operate also on pre-loaded TCGA curated data. We also deployed the GMQL execution engine as a docker file, available on Docker Hub (https://hub.docker.com/) as *gecopolimi/gmql*; doing so, we could integrate GMQL also in the FireCloud cloud-based analysis service offered by the Broad Institute. There, we provide a set of example methods and configurations which add a GMQL task as the last step of a FireCloud pipeline including mutation calling implemented in GATK, and a simpler task which performs the query of Section 2. These examples are documented and designed to facilitate the use of GMQL by the FireCloud users.

6 Discussion

Being a declarative language, GMQL enables users to specify and run operations, even complex, on genomic data, even big and heterogeneous, using a few high-level constructs, thereby allowing users to focus on the analytical goals rather than on their technical details. Although the GMQL syntax is similar to the Structured Query Language one, the compact and nested form of SQL is found to be less easy to use and to learn when compared with the progressive style of building results using sequences of operations, each focused on a specific data transformation. We agree with (Olston *et al*., 2008) that to programmers the latter "method is much more appealing than encoding their task as an SQL query".

The innovations that we introduced in developing the new GMQL system are summarized as follow, together with the achieved advantages:

- Well-designed system architecture with modular organization; system modules are easily tested, maintained or replaced;
- Intermediate, language-independent representation (in form of DAG), whose nodes are mapped to any target implementation technology, for supporting different implementations (e.g., using Spark, Flink, or SciDB engines) for several target systems (local or remote);

- High-level, technology-independent data repository abstraction for supporting different repository technologies (e.g., local file system, Hadoop File System, database, or others) by simply implementing the repository interface;
- Web and API interfaces to the language, enabling both its friendly use and the embedding of high-level data extraction and processing operations within classical languages for data science, allowing the seamless integration of data exploration and data analysis.

Several aspects of the new GMQL system architecture have been designed for fast execution on big datasets. The use of cloud computing technologies brings parallelism efficiency through data distribution (using Hadoop Distributed File System) and processing distribution (using Apache Spark or other engines). In order to better support parallelism w.r.t. native parallelism of cloud engines, we proposed and adopted new binning algorithms suitable for genomic data (Kaitoua *et al*., 2017; Cattani *et al*., 2017a); we compared GMQL prototypical operations on different engines (Spark vs. Flink and Spark vs. SciDB), as detailed elsewhere (Bertoni *et al*., 2015, Cattani *et.al.* 2017b), and thereby we also learnt about optimal parameter setting for genomic query processing.

The new GMQL system design is inspired by dominant cloud computing paradigms, which are supported by a variety of next-generation cloud-based data engines. GMQL scripts are translated into such paradigms and then executed; thus, the evolution of GMQL (in terms of portability, performance, scalability) will be well-supported by the key actors of cloud computing.

In conclusion, the new GMQL system is an easy-to-use, versatile and interoperable resource for processing big heterogeneous genomic datasets in order to extract candidate targets for biomedical knowledge discovery, as demonstrated by many provided examples.

Acknowledgement

We thank Daniele Braga and Alessandro Campi for their collaboration to the GeCo project and for their critical evaluation of design decisions for the new GMQL system.

Funding

This work was supported by the ERC Advanced Grant 693174 “Data-Driven Genomic Computing (GeCo)”.

Conflict of Interest: none declared.

References

Albrecht,F. *et al*. (2016) DeepBlue epigenomic data server: programmatic data retrieval and analysis of epigenome region sets. *Nucleic Acids Res*., **44**(W1), W581-W586.

Bernasconi,A. *et al*. (2017) Conceptual modeling for genomics: Building an integrated repository of open data. In Mayr,H. *et al*. (eds.), *Proc. 36th Int. Conf. on Conceptual Modeling (ER 2017)*. Springer Verlag, Berlin, D, pp. 325-339. LNCS vol 10650.

Bernstein,B.E. *et al*. (2010) The NIH Roadmap Epigenomics Mapping Consortium. *Nat. Biotechnol*., **28**(10), 1045-1048.

Bertoni,M. *et al*. (2015) Evaluating cloud frameworks on genomic applications. In Ho,H. *et al*. (eds.), *Proc. IEEE Int. Conf. Big Data*. IEEE Computer Society, Washington, DC, pp. 193-202.

Brown,P.G. (2010) Overview of SciDB: large scale array storage, processing and analysis. In Mokbel,M. (ed.), *Proc. 2010 ACM SIGMOD Int. Conf. on Management of Data*. ACM, Chicago, IL, pp. 963-968.

Cancer Genome Atlas Research Network *et al*. (2013) The Cancer Genome Atlas Pan-Cancer analysis project. *Nat. Genet*., **45**(10), 1113-1120.

Cattani,S. *et al*. (2017a) Bi-dimensional binning for big genomic datasets. In Koutris,P. (ed.), *Proc. 4th Workshop on Algorithms and Systems on MapReduce and Beyond, ACM*, Chicago, IL, pp. 9:1-9:4.

Cattani,S. *et al*. (2017b) Evaluating big data genomic applications on SciDB and Spark. In Al-Tawil,M. *et al*. (eds.), *Proc. Int. Conf. Web Eng.* Springer Verlag, Berlin, D, pp. 482-493.

Cingolani,P. *et al*. (2015) BigDataScript: a scripting language for data pipelines. *Bioinformatics*, **31**(1), 10-16.

Cumbo,F. *et al*. (2017) TCGA2BED: extracting, extending, integrating, and querying The Cancer Genome Atlas. *BMC Bioinformatics*, **18**(1), 6.

Dean,J. and Ghemawat,S. (2010) MapReduce: A flexible data processing tool. *Commun. ACM*, **53**(1), 72-77.

Del Chierico,F. *et al*. (2015) Choice of next-generation sequencing pipelines. *Methods Mol. Biol*., **1231**, 31-47.

ENCODE Project Consortium. (2012) An integrated encyclopedia of DNA elements in the human genome. *Nature*, **489**(7414), 57-74.

Eric,E. *et al*. (2010) Computational solutions to large-scale data management and analysis. *Nat. Rev. Genet*. **11**(9), 647-657.

Freese,N.H. *et al*., (2016) Integrated genome browser: visual analytics platform for genomics. *Bioinformatics*, **32**(14), 2089-2095.

Goodwin,S. *et al*. (2016) Coming of age: ten years of next-generation sequencing technologies. *Nat. Rev. Genet*., **17**(6), 333-351.

Guðbjartsson,H. *et al*. (2016) GORpipe: a query tool for working with sequence data based on a Genomic Ordered Relational (GOR) architecture. *Bioinformatics*, **32**(20), 3081-3088.

Huber,W. *et al*. (2015) Orchestrating high-throughput genomic analysis with Bioconductor. *Nat. Methods*, **12**(2), 115-121.

Kaitoua,A. *et al*. (2017) Framework for supporting genomic operations. *IEEE Trans. Comput*., **66**(3), 443-457.

Karolchik,D. *et al*. (2012) The UCSC Genome Browser. *Curr. Protoc. Bioinformatics*, **Chapter 1**(Unit1), 4.

Kozanitis,C. *et al*. (2014) Using Genome Query Language to uncover genetic variation. *Bioinformatics*, **30**(1), 1-8.

Jalili,V *et al*. (2017) Explorative visual analytics on interval-based genomic data and their metadata. *BMC Bioinformatics*, **18**(1), 536.

Masseroli,M. *et al*. (2015) GenoMetric Query Language: A novel approach to large-scale genomic data management. *Bioinformatics*, **31**(12), 1881-1888.

Masseroli,M. *et al*. (2016) Modeling and interoperability of heterogeneous genomic big data for integrative processing and querying. *Methods*, **111**, 3-11.

McKenna,A. *et al*. (2010) The Genome Analysis Toolkit: MapReduce framework for analyzing next-generation DNA sequencing data. *Genome Res*., **20**(9), 1297-1303.

Mrozek,D *et al*. (2014) Cloud4Psi: cloud computing for 3D protein structure similarity searching. *Bioinformatics*, **30**(19), 2822-2825.

Mrozek,D. *et al*. (2016) HDInsight4PSi: Boosting performance of 3D protein structure similarity searching with HDInsight clusters in Microsoft Azure cloud. *Information Sciences*, **349-350**, 77-101.

Neph,S. *et al*. (2012) BEDOPS: high-performance genomic feature operations. *Bioinformatics*, **28**(14), 1919-1920.

O'Driscoll,A. *et al*. (2013) 'Big data', Hadoop and cloud computing in genomics. *J. Biomed. Inform*., **46**(5), 774-781.

Olston,C. *et al*. (2008) Pig Latin: A not-so-foreign language for data processing. In Lakshmanan,L.V.S. *et al*. (eds.), *Proc. 2008 ACM SIGMOD Int. Conf. on Management of Data*. ACM, New York, NY, pp. 1099-1110.

Ovaska,K. *et al*. (2013) Genomic Region Operation Kit for extensible processing of deep sequencing data. *IEEE/ACM Trans. Comput. Biol. Bioinform*., **10**(1), 200-206.

Quinlan,A.R. and Hall,I.M. (2010) BEDTools: a flexible suite of utilities for comparing genomic features. *Bioinformatics*, **26**(6), 841-842.

Robinson,J. *et al*. (2011) Integrative genomics viewer. *Nat. Biotechnol*., **29**(1), 24-26.

Roy,A. *et al*. (2017) Massively parallel processing of whole genome sequence data: an in-depth performance study. In Chirkova,R. and Yang,J. (eds.), *Proc. 2017 ACM Int. Conf. on Management of Data*. ACM, Chicago, IL, pp. 187-202.

Schmidt,B. and Hildebrandt,A. (2017) Next-generation sequencing: big data meets high performance computing. *Drug Discov. Today*, **22**(4), 712-717.

Shvachko,K. *et al*. (2010) The Hadoop distributed file system. In Khatib,M.G. (ed.) *Proc. 2010 IEEE 26th Symp. Mass Storage Systems and Technologies (MSST)*. IEEE Computer Society, Washington, DC, pp. 1-10.

Siva,N. (2015) UK gears up to decode 100,000 genomes from NHS patients. *Lancet*, **385**(9963), 103-104.

Weiwiorka,M.S. *et al*. (2014) SparkSeq: Fast, scalable and cloud-ready tool for the interactive genomic data analysis with nucleotide precision. *Bioinformatics*, **30**(18), 2652-2653.

Zhu,X. *et al*. (2017) START: a system for flexible analysis of hundreds of genomic signal tracks in few lines of SQL-like queries. *BMC Genomics*, **18**(1), 749.