Bioinformatics, 31(12), 2015, 1881–1888
doi: 10.1093/bioinformatics/btv048
Advance Access Publication Date: 3 February 2015
Original Paper

OXFORD

Genome analysis

GenoMetric Query Language: a novel approach to large-scale genomic data management

Marco Masseroli[1,*], Pietro Pinoli[1], Francesco Venco[1], Abdulrahman Kaitoua[1], Vahid Jalili[1], Fernando Palluzzi[1], Heiko Muller[2] and Stefano Ceri[1]

[1]Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, 20133, Milan and [2]Center for Genomic Science of IIT@SEMM, Istituto Italiano di Tecnologia (IIT), 20139 Milan, Italy

*To whom correspondence should be addressed.
Associate Editor: Inanc Birol

Received on October 20, 2014; revised on January 4, 2015; accepted on January 21, 2015

Abstract

Motivation: Improvement of sequencing technologies and data processing pipelines is rapidly providing sequencing data, with associated high-level features, of many individual genomes in multiple biological and clinical conditions. They allow for data-driven genomic, transcriptomic and epigenomic characterizations, but require state-of-the-art 'big data' computing strategies, with abstraction levels beyond available tool capabilities.
Results: We propose a high-level, declarative GenoMetric Query Language (GMQL) and a toolkit for its use. GMQL operates downstream of raw data preprocessing pipelines and supports queries over thousands of heterogeneous datasets and samples; as such it is key to genomic 'big data' analysis. GMQL leverages a simple data model that provides both abstractions of genomic region data and associated experimental, biological and clinical metadata and interoperability between many data formats. Based on Hadoop framework and Apache Pig platform, GMQL ensures high scalability, expressivity, flexibility and simplicity of use, as demonstrated by several biological query examples on ENCODE and TCGA datasets.
Availability and implementation: The GMQL toolkit is freely available for non-commercial use at http://www.bioinformatics.deib.polimi.it/GMQL/.
Contact: marco.masseroli@polimi.it
Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 Introduction

Next generation sequencing (NGS) allows high-throughput genome sequencing (DNA-seq), transcriptome profiling (RNA-seq), DNA–protein interaction assessment (ChIP-seq) and epigenome characterization (ChIP-seq, BS-seq, DNase-seq). Continuous improvements of NGS technologies in quality, cost of results (http://www.genome.gov/sequencingcosts/) and sequencing time are leading shortly to the possibility of sequencing an entire human genome in few minutes for a cost of <$1000 (Hayden, 2014; Sheridan, 2014). Sequencing of genomes in several different biological and clinical conditions for their genomic, transcriptomic and epigenomic characterization is becoming widespread.

Very large-scale sequencing projects are emerging. The *1000 Genomes Project* aims at establishing an extensive catalogue of human genomic variation (1000 Genomes Project Consortium *et al.*, 2010); it recently released a list of more than 79 million variant sites, using data from 2535 individuals from 26 different populations (http://www.1000genomes.org/announcements/initial-phase-3-variant-list-and-phased-genotypes-2014-06-24/). *The Cancer Genome Atlas* (TCGA) project is a full-scale effort to systematically explore the

entire spectrum of genomic changes involved in human cancer (Cancer Genome Atlas Research Network *et al*., 2013). Through its data portal (https://tcga-data.nci.nih.gov/tcga/), it offers access to clinical information as well as genomic characterization of the tumor genomes of more than 11 000 cases regarding about 30 different cancer types. The *Encyclopedia of DNA elements* (ENCODE) project (ENCODE Project Consortium, 2012) provides public access to more than 4000 experimental datasets, including the just released data from its Phase 3: more than 760 experiments of mainly ChIP-seq and RNA-seq assays in human and mouse (https://www.encodedcc.org/).

Thanks to such high availability of several different types of NGS data from numerous individual genomes, it is now possible to look at multiple instances of many different genomic features simultaneously, so as to characterize their functional role and elucidate genetic and epigenetic phenomena. This requires a new generation of informatics systems and languages for querying heterogeneous datasets; it also requires the state-of-the-art parallel computing strategies for distributing the computation upon clouds of computing elements, to achieve scalability and performance.

Although most of current bioinformatics analysis tools do not support parallelization, lately cloud computing and high-performance parallel systems, mainly based on *Hadoop* (Shvachko *et al*., 2010) and *MapReduce* (Dean and Ghemawat, 2010) frameworks, which are typically used in other areas, have also been adopted in bioinformatics (O'Driscoll *et al*., 2013; Taylor, 2010; Zou *et al*., 2014). A few specific parallel tools have been implemented (McKenna *et al*., 2010; Nordberg *et al*., 2013; Schumacher *et al*., 2014; Weiwiorka *et al*., 2014). In particular, *BioPig* (Nordberg *et al*., 2013) and *SeqPig* (Schumacher *et al*., 2014) have been proposed for the efficient processing of bioinformatics programs using *Pig Latin* (Olston *et al*., 2008), a high-level language for batch data processing. Recently, also *SparkSeq* (Weiwiorka *et al*., 2014) has been proposed as general purpose tool for processing of DNA and RNA sequencing data using the *Apache Spark* (Zaharia *et al*., 2012) engine.

In this article, we propose GenoMetric Query Language (GMQL), with the associated data model and data management system, to query and compare multiple and heterogeneous genomic datasets for biomedical knowledge discovery. GMQL uses the Genomic Data Model (GDM) based on the notion of *genomic region,* which may span several bases. Each region can be compared with millions of other regions, typically using metric properties (this justifies the language name); in addition, GDM also covers *metadata* of arbitrary structure. GMQL is a high-level, declarative language that allows expressing queries easily 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; thus, it does not require software programmer expertise. From a system perspective, GMQL is part of *GenData 2020* (http://www.bioinformatics.deib.polimi.it/gendata/), a system, which supports a data warehouse layer for storing data files in their original formats, providing suitable privacy levels, and a data selection layer, enabling abstractions from file formats and parallelization.

The work in Kozanitis *et al*. (2014) is the closest to ours, as it reports a Genome Query Language (GQL) for NGS data using an SQL extension. Our approach exhibits two main differences with respect to GQL and to all mentioned previous works. First and foremost, they start from the reads of NGS machines, i.e. raw data, and thus they must address a number of problems for identifying regions of interest on the genome, which in the current practice are normally solved by *ad hoc* specific tools (e.g. mutation and peak callers); *conversely, GMQL starts from processed datasets*. Processing raw data from within an integrated system is potentially very powerful, as no information is left outside of the query system; but there is as well a risk of replicating efforts and of giving up the efficiency of specialized data analysis tools that are widely accepted by the scientific community.

A second major difference with previous works is that none of them integrate metadata within their computation; *conversely, GMQL supports metadata management*. Thus, it can use metadata for sample selection and matching, and query results carry along metadata that can be inspected after query execution. Many widely available experimental data (e.g. in ENCODE and TCGA) provide processed datasets and metadata; thus, GMQL can directly be used on them. In addition, each research laboratory produces data in many processed formats; through GMQL they can be compared with both locally produced and publicly available data. In a mid-term perspective, GMQL will be the appropriate tool for querying millions of processed genomic datasets and samples that will become available.

2 Results

2.1 Genomic data model

GDM describes genomic samples by means of two fundamental abstractions, one for genomic regions and one for their metadata. The former one characterizes the sample regions with their coordinates, relative to a reference genome, and a free number of structured, high-level properties [e.g. variation from a reference sequence, level of expression or protein binding enrichment ('peak') characteristics such as peak statistical or geometric properties]. The latter one defines the metadata of a sample (e.g. experimental conditions, biological specimen features, cell line or the patient phenotypes when data have clinical nature).

The key aspect of the model is the notion of genomic region; a genomic region r_i is a well-defined portion of a genome, qualified by a quadruple of values called region coordinates:

$$r_i =< chr, left, right, strand >,$$

where *chr* is the genome chromosome name, *left* and *right* are the two ends of the region along the chromosome coordinates, *strand* represents the DNA strand (i.e. the direction of DNA reading), encoded as either + or −, and can be missing (encoded as *). According to the UCSC notation (http://genome.ucsc.edu/FAQ/FAQtracks.html#tracks1), we use the 0-based, half-open inter-base coordinates, i.e. the considered genomic sequence is [*left*, *right*). Thus, a region r_i corresponds to all the nucleotides whose position is between its *left* and *right* ends; however, in general, we do not include nucleotide sequences within region data, but rather we store high-level properties of the region (e.g. for ChIP-seq peak samples, the peak's *p*-value and *q*-value).

Metadata describe the experimental, biological and clinical properties associated with each genomic data sample; due to the great heterogeneity of information that can be associated with a data sample, in the GDM they are represented as arbitrary attribute-value pairs. We expect metadata to include at least the experiment type or the sequencing and analysis method used for the data production, the experimental condition (e.g. antibody target), the sequenced organism and tissue or cell line; in case of clinical study, individual's descriptions including phenotypes.

Formally, in the GDM a sample *s* is defined as a triple:

$$s =< id, \{< r_i, v_i >\}, \{m_j\} >,$$

where *id* is the sample identifier (of type long); each pair $< r_i, v_i >$ represents a region, with coordinates r_i (the four fixed attributes *chr*,

left, *right* and *strand* of type string, integer, integer and string, respectively) and values v_i (which are typed attributes; we assume attribute names of a sample to be different and their types to be of any elementary type, i.e. Boolean, integer, long, float or string); m_j are attribute-value pairs, with values of type string. Thus, each sample s has specific attributes describing its region properties and an associated set of attribute-value pairs referred to as the metadata of s. The region data schema of s is the set of all attribute names used for region coordinates and values, and the region data type of s is the record of all the elementary types of the corresponding attributes. The use of a type system to express region data allows for controlled arbitrary operations upon type-compatible values.

In GDM, a dataset is a collection of samples with the same region data schema and type, which contain region values compliant with the data schema and a sample identifier that is unique within each dataset; thus, datasets are homogeneous collections of samples, which are typically produced within the same project (either at a genome research center, or within an international consortium) using the same technology and tools. Datasets are represented using two normalized data structures, one for region data and one for metadata; an example representing a dataset of ChIP-seq experiments is shown in Tables 1 and 2. Note that the sample *id* attribute (first column in each data structure) provides a many-to-many connection between regions (Table 1) and metadata (Table 2) of a sample (e.g. in Tables 1 and 2, sample 1 has four regions and six metadata attributes, whereas sample 2 has three regions and five metadata attributes). Through the use of a data type system to express region data and of arbitrary attribute-value pairs for metadata, GDM provides interoperability across datasets produced by different experimental techniques.

2.2 GenoMetric Query Language

GMQL is a high-level language inspired by classic traditions of database management (Edward T. Codd's relational algebra dating 1971), which aims at substantially changing the level of interaction between biologists and NGS data. It extends conventional algebraic operations with bioinformatics domain-specific operations specifically designed for genomics; thus, it supports knowledge discovery across thousands or even millions of samples, both for what concerns regions that satisfy biological conditions and their relationship to experimental, biological or clinical metadata.

The name 'GenoMetric Query Language' stems from the language's ability to deal with genomic distances, which are measured as nucleotide bases between genomic regions (aligned to the same reference) and computed using arithmetic operations between region coordinates. GMQL is inspired by the Pig Latin language (Olston *et al.*, 2008), which combines high-level declarative style in the spirit of SQL with the low-level procedural form of MapReduce (Dean and Ghemawat, 2010); yet, it is much simpler and demands much less informatics expertise.

Table 1. Excerpt of region data of a dataset with two ChIP-seq samples

id	chr	left	right	strand	*p*-value
1	2	2476	3178	*	0.00000000200
1	2	15 235	15 564	*	0.00000000052
1	5	8790	11 965	*	0.00000000009
1	5	75 980	76 342	*	0.00000000037
2	16	862	923	*	0.00000000018
2	16	1276	1409	*	0.00000000006
2	20	3852	4164	*	0.00000000031

Corresponding sample metadata are in Table 2.

Table 2. Excerpt of metadata of a dataset with two ChIP-seq samples

id	attribute	value
1	antibody_target	CTCF
1	cell	HeLa-S3
1	cell_karyotype	cancer
1	cell_organism	human
1	dataType	ChipSeq
1	view	Peaks
2	antibody_target	JUN
2	cell	H1-hESC
2	cell_organism	human
2	dataType	ChipSeq
2	view	Peaks

Corresponding sample region data are in Table 1.

A GMQL query (or program) is expressed as a sequence of GMQL operations with the following structure:

$$< variable >= operator(< parameters >) < variables >,$$

where each variable stands for a GDM dataset. Operators apply to one or more operand variables and construct one result variable; parameters are specific for each operator. Parameters of several operators include predicates, used to select and join samples; predicates are built by arbitrary Boolean expressions of simple predicates, as it is customary in relational algebra. Predicates on region data must use attributes in the region's data schema; predicates on metadata may use arbitrary attributes. Thus, when a predicate on region data uses an illegal attribute, the query is also illegal; when a predicate on metadata uses an attribute not present in the metadata, the predicate is unknown. GMQL operators form a closed algebra; hence, operator results are expressed as new datasets derived from their operands and from the operator specifications.

The language supports a very rich set of predicates describing distal conditions, i.e. distal properties of regions (e.g. being among the regions at minimal distance above a given threshold from given genes). Furthermore, the management of stranded regions (i.e. regions with an orientation) is facilitated by predicates that deal with such orientation (e.g. to locate the region's starts and stops according to the strand, or the upstream or downstream directions with respect to the region's ends). GMQL includes operators on metadata (*SELECT*, *AGGREGATE*, *ORDER*) and regions (*PROJECT*, *COVER*, *SUMMIT*), as well as operators on multiple datasets (*UNION*, *DIFFERENCE*, *JOIN*, *MAP*); the full description and language specification of GMQL is provided at the GMQL web site http://www.bioinformatics.deib.polimi.it/GMQL/. A typical GMQL query starts with a *SELECT* operation, which creates a dataset with only the data samples that it filters out from an input dataset by using a predicate upon their metadata attributes (the input dataset can be any available data collection in a standard data format, such as BED, bedGraph, broadPeak, narrowPeak, VCF, GTF). Then, the query proceeds by processing the selected samples in batch with operations applied on their region data and/or metadata. Finally, it ends with a *MATERIALIZE* operation, which stores a dataset by saving the region data of each of its samples in an individual text file in standard GTF format and the related metadata in an associated tab delimited text file.

We note a fundamental difference between GMQL and other available systems, namely the ability to express operations over datasets with thousands of samples in a compact way, by using few

lines of abstract, high-level code (which implicitly express iteration over samples). This is very different from the current practice of embedding data management operations within the code of programming language scripts (e.g. Python, JavaScript or Java).

3 Examples

We demonstrate the power and flexibility of GMQL by presenting some typical queries that show GMQL at work in a rich set of biological use case examples. GMQL shows its assets in particular when it is applied on numerous samples containing many regions and of multiple data types, in order to identify their genomic regions that satisfy given distance constraints. At http://www.bioinformatics.deib.polimi.it/GMQL/ we provide an easy to install toolkit for stand-alone use of GMQL, together with the below GMQL example queries and small-scale datasets just for example testing. Note that GMQL performs worse than some other available systems on a small number of small-scale datasets, but these other systems are not cloud-based; hence, they are not adequate for efficient big data processing and, in some cases, they are inherently limited in their data management capacity, as they only work as RAM memory resident processes.

3.1 Example 1: find ChIP-seq peaks in promoter regions

"*In each enriched region (often called 'peak') sample from ENCODE human ChIP-seq experiments, find how many peaks co-occur in each promoter region*".

```
HM_TF = SELECT(dataType == 'ChipSeq' AND
   view == 'Peaks') HG19_PEAK;
TSS = SELECT(ann_type == 'TSS' AND
   provider == 'UCSC') HG19_ANN;
PROM = PROJECT(score >= 1000; start = start - 2000,
   stop = stop + 1000) TSS;
PROM_HM_TF = MAP(COUNT) PROM HM_TF;
MATERIALIZE PROM_HM_TF;
```

This GMQL query starts by selecting all the ChIP-seq peak samples from the dataset *HG19_PEAK*, which can represent, e.g., all the ENCODE peak samples called from human NGS experiments aligned to the human reference genome assembly 19 (https://genome.ucsc.edu/ENCODE/dataMatrix/encodeChipMatrixHuman.html). From an annotation base for the same human genome assembly (represented by the dataset *HG19_ANN)*, it also selects the transcription start sites (TSS), provided by the UCSC database (https://genome.ucsc.edu/cgi-bin/hgTables) from SwitchGear Genomics (http://switchgeargenomics.com/) and defines the promoter regions as the ones extending from 2000 bases upstream a TSS to 1000 bases downstream of the same TSS (using only TSS with high confidence score based on existing experimental evidence). Finally, it maps the peaks of each sample to the promoter regions and, for each promoter, counts how many of such peaks co-localize with the promoter, saving the obtained results.

Although very simple, this example shows the power of the Map operation, which easily relates experimental data to known annotations. Note that a Select-Project-Map sequence of GMQL operations is comparable with the basic Select-From-Where sequence of SQL statements. When applied to the entire ENCODE data collection, at the time of writing this GMQL query selects 2423 samples including a total of 83 899 526 peaks, which are mapped to 131 780 promoters, saving as result 29 GB of data files in standard GTF format. Executed in a Hadhoop framework on a single server equipped with a Dual Intel Xeon ES-2650 processor, 64 GB of RAM and 16 TB of disks (RAID 5, 5 disks), it required 18 min and 33 s.

3.2 Example 2: find distal bindings in transcription regulatory regions

"*Find all enriched regions (peaks) in CTCF transcription factor (TF) ChIP-seq samples from different human cell lines which are the nearest regions farther than 100 kb from a transcription start site (TSS). For the same cell lines, find also all peaks for the H3K4me1 histone modifications (HM) which are also the nearest regions farther than 100 kb from a TSS. Then, out of the TF and HM peaks found in the same cell line, return all TF peaks that overlap with both HM peaks and known enhancer (EN) regions*".

```
TF = SELECT(dataType == 'ChipSeq' AND view == 'Peaks'
   AND antibody_target == 'CTCF') HG19_PEAK;
HM = SELECT(dataType == 'ChipSeq' AND view == 'Peaks'
   AND antibody_target == 'H3K4me1') HG19_PEAK;
TSS = SELECT(ann_type == 'TSS' AND
   provider == 'UCSC') HG19_ANN;
EN = SELECT(ann_type == 'enhancer' AND
   provider == 'UCSC') HG19_ANN;
TF1 = JOIN(first(1) after distance 100 000, right_distinct)
   TSS TF;
HM1 = JOIN(first(1) after distance 100 000, right_distinct)
   TSS HM;
HM2 = JOIN(distance < 0, int) EN HM1;
TF_res = JOIN(left->cell == right->cell, distance < 0,
   right_distinct) HM2 TF1;
MATERIALIZE TF_res;
```

This second example, whose context is illustrated in Figure 1, shows that GMQL is a powerful expressive language to answer frontier epigenomics questions.

The GMQL query selects TF and HM enriched region samples from *HG19_PEAK* [a collection of experimental data files of signal enrichment called regions from NGS human samples aligned to the human genome assembly 19, e.g. provided by ENCODE (https://genome.ucsc.edu/ENCODE/dataMatrix/encodeChipMatrixHuman.html)]. It also selects TSS and EN known human annotation regions from *HG19_ANN* [e.g. provided by the UCSC database (https://genome.ucsc.edu/cgi-bin/hgTables) from SwitchGear Genomics (http://switchgeargenomics.com/) and Vista Enhancer (http://enhancer.lbl.gov/), respectively]. Then, for each sample, it computes in *TF1* the TF regions that are at minimal distance from a TSS, provided that such distance is greater than 100 000 bases, and in *HM1* the HM regions that are also at minimal distance from a TSS, with the same constraint. Note that, in addition to the distal condition, the Join parameter also indicates that the result must include only the right sample matching regions, for *TF1* and *HM1* respectively.

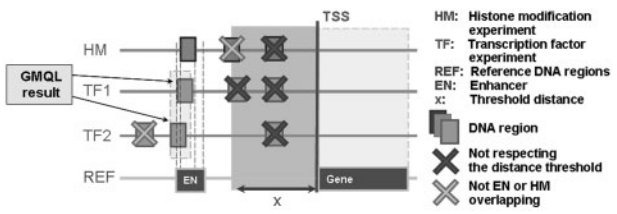

Fig. 1. Representation of the histone modification (HM) and transcription factor (TF) binding site enriched regions ('peaks'), known reference DNA regions and their distance relationships involved in Example 2

Next, in *HM2* the query computes those *HM1* regions that intersect with enhancer regions in *EN*, by taking the intersection of resulting regions. Finally, in *TF_res*, which is then saved as a result, for each cell line, it computes the TF regions in *TF1* that intersect with regions in *HM2*, also this time taking only the right sample matching regions. At the time of writing, all enriched region samples from ENCODE human ChIP-seq amount to 2423 samples from 120 cell lines, which become 987 samples after merging sample replicates. When applied to such merged replicate samples, initially this example query selects 90 CTCF and 21 different H3K4me1 samples from 93 cell lines, including a total of 3 317 361 peaks; finally, as the result it finds 42 CTCF peaks in 11 samples from 11 cell lines.

The Map operation (in Examples 1, 3 and 4) as well as the Join operation with *distance* and *first after distance* functions (in Example 2) highlight the power of GMQL in performing genometric evaluations in batch on multiple samples at a time; they are normally performed by executing data manipulation scripts, developed by individual researchers in different programming languages. Some tools provide libraries for supporting genometric queries (Neph *et al*., 2012; Ovaska *et al*., 2013; Quinlan and Hall, 2010), but they are not intended for large-scale querying.

3.3 Example 3: associating transcriptomics and epigenomics

"*In RNA-seq experiments of human cancer cell line HeLa-S3, find the average expression of each exon. Then, in ChIP-seq experiments of the same cell line, find the average signal of H3K4me3 histone modifications within each exon*".

```
Exon = SELECT(ann_type == 'exons' AND
   provider == 'RefSeq') HG19_ANN;
HM = SELECT(dataType == 'ChipSeq' AND view == 'Peaks'
   AND cell == 'HeLa-S3' AND antibody_target == 'H3K4me3')
   HG19_PEAK;
RNA = SELECT(dataType == 'RnaSeq' AND
   view == 'ExonsDeNovo' AND cell == 'HeLa-S3') HG19_RNA;
RNA2 = PROJECT(iIDR < 0.05;
   signal AS RPKM1 / 2 + RPKM2 / 2) RNA;
Exp = UNION RNA2 HM;
Genome_space = MAP(signal_avg AS AVG(signal)) Exon Exp;
MATERIALIZE Genome_space;
```

This third example shows that datasets produced by different experiment types, such as RNA-seq and ChIP-seq, can be used in the same GMQL query, thanks to the interoperability provided by GDM; the *UNION* operator takes care of automatically unifying their different region data schemas. For each RNA-seq sample, the signal in exons with reproducible expression [i.e. with Irreproducibility Discovery Rate (IDR) < 0.05] is averaged (in the new attribute *signal*) over two replicates (ENCODE provides RNA-seq data files with expression data quantified for two replicates separately as RPKM, i.e. reads per kilobase of exon per million reads mapped). Then, the Map operation averages, within each exon, the signal in each RNA-seq or ChIP-seq sample and constructs a query result, called *Genome Space*, which is the ideal starting point for subsequent data analysis steps; they are not covered by GMQL, but can be performed using classical tools, which are immediately applicable to GMQL output. For instance, these analyses may include building lists of top k genes with most similar expression to that of a given gene, or identifying groups of genes and their associated histone modifications and transcription factors with similar signal patterns, e.g. by using bi-clustering.

3.4 Example 4: find somatic mutations in exons

"*Consider mutation data samples of human breast cancer cases. For each sample, quantify the mutations in each exon and select the exons with at least one mutation. Return the list of samples ordered by the number of such exons* ".

```
Mut = SELECT(dataType == 'DNASeq' AND
   tumor_tissue_site == 'Breast') HG19_MUT;
Exon = SELECT(ann_type == 'exons' AND
   provider == 'RefSeq') HG19_ANN;
Exon1 = MAP(mut_count AS COUNT) Exon Mut;
Exon2 = PROJECT(mut_count >= 1) Exon1;
Exon3 = AGGREGATE(exon_count AS COUNT) Exon2;
Exon_res = ORDER(DESC exon_count) Exon3;
MATERIALIZE Exon_res;
```

This fourth example shows that GMQL is very effective at counting, in batch on multiple samples, mutations that are mapped upon known regions (in this case exons) and extracting those regions having more mutations than a given threshold, ordering samples according to their number of regions extracted. Known human protein-coding and non-protein-coding exon regions are selected from *HG19_ANN*; such regions are provided by the UCSC database (https://genome.ucsc.edu/cgi-bin/hgTables), after retrieving them from the NCBI RNA reference sequences collection (RefSeq). Count of data sample items is performed by the Map and Aggregate operations. The Map counts mutations in each sample within each exon while mapping the mutations to the exon regions; after removal of the exons in each sample that do not contain mutations, the Aggregate counts how many exons remain in each sample and stores the result in the sample metadata as a new attribute–value pair. Note that, also in this example, the query is applied in batch on multiple data samples, i.e. all those samples selected from the *HG19_MUT* collection, which can be very many (in the case of the publicly available TCGA data, at the time of writing the available breast cancer patient samples were 963 for a total of 87 131 mutations). By applying this GMQL query to the curated somatic mutation data publicly available in TCGA from breast cancer patients and considering all 482 313 exon regions of 46 949 human protein-coding and non-protein-coding genes provided by RefSeq, at the time of writing it extracted 963 breast cancer patient samples with mutations involving 54 688 distinct exons of 41 364 genes.

4 Discussion

Our system supports efficient high-level query processing of thousands of experimental data samples, produced with a variety of experimental methods and encoded in a variety of data formats, together with their biological and clinical metadata descriptions as well as multiple annotation data. Focused on assisting knowledge extraction, it is meant to operate on higher level data obtained after raw data preprocessing and feature calling, rather than on raw data directly. This offers the advantage of not interfering with the variety of data preprocessing tools and pipelines that are already in place in the different research centers, as well as of directly benefitting of their output, thanks to the interoperability and data integration support ensured by the data model and the standard data formats used. Thus, it also allows leveraging the high-level data, including variant calling, gene expression and region enrichment (i.e. peak) calling

data, which are increasingly available within public large data collections (e.g. 1000 Genomes Project, TCGA and ENCODE); public data can be used not only by themselves, but also together with in house produced experimental data, for integrated evaluations and comparisons with increased support.

The rich variety of provided biological use case examples, which are based on public datasets from relevant data collections such as ENCODE and TCGA, proves the declarative expressivity, power, flexibility and ease of use of GMQL, as well as its effectiveness in support of answering typical relevant biological questions on massive data. Although several tools for genomic data processing have been proposed, none of them encompasses all the GMQL features and none can perform all those evaluations, at least not with the simplicity and power of GMQL.

As clarified in 'Methods' section, another important aspect of GMQL is that the language design was also inspired by dominant cloud computing paradigms, such as the *Apache Pig* platform and *Lucene* indexing, which are supported in Hadoop and by a variety of next-generation cloud-based systems. GMQL queries are translated into such paradigms and then executed. Thus, the evolution of GMQL (in terms of performance, portability, scalability) is very well-supported by the key actors of cloud computing.

BEDTools (Quinlan and Hall, 2010) and BEDOPS (Neph *et al*., 2012) easily and efficiently process region data, but of individual samples (at most two at a time, for their comparison), requiring verbose scripts for multiple sample processing, with lower performance. Furthermore, they cannot manage sample metadata and do not have an internal standard for data format; this may lead different scripts to generate different output formats, with the need for external languages (e.g. AWK or SED) to manipulate their outputs, which increases the verbosity of their scripts (see Supplementary material for thorough comparison).

GROK (Ovaska *et al*., 2013) supports abstractions, but of lower-level than GMQL, and some low-level operations which are not directly supported by GMQL (e.g. region flipping); in contrast, GMQL supports high-level declarative operations, such as Join and Map. Furthermore, GROK does not support sample metadata and big data processing, being unsuitable for parallelization.

GQL (Kozanitis *et al*., 2014) offers a declarative query language for genomics, but limited to variant calling, starting from DNA-seq aligned data. As well as all the other tools, it cannot deal with metadata describing experimental, biological or clinical characteristics of the data samples; however, it is amenable to parallelization, as any SQL-like language.

Some other tools for parallel processing of genomic data have been developed [e.g. GATK (McKenna *et al*., 2010), SAMtools (Li *et al*., 2009), BioPig (Nordberg *et al*., 2013)], but all of them deal with raw or aligned data; thus, they focus on data preprocessing or feature calling, without taking advantage of the quickly increasing amount of processed data which are available for knowledge discovery. BioPig provides a set of Pig Latin extensions for specific processing of data files produced by NGS machines. In BioPig, a software developer can write user defined functions (UDFs) in Java programming language, but then s/he has to manage the Pig Latin scripts manually.

We stress that GMQL is designed for cloud computing processing of big data in the Hadoop framework. This makes it suitable also for remotely running in the cloud, using web services and appropriate software frameworks like Galaxy, which allows for the pipelining/integration of multiple analysis tools (Giardine *et al*., 2005; Goecks *et al*., 2010); this avoids the need for extensive infrastructure on the user's part and allows taking advantage of the genomic data already available in the cloud, e.g. the 1000 Genomes Project data available on Amazon Elastic Compute Cloud (http://aws.amazon.com/datasets/4383).

GMQL can be used also with small data and on non-parallel computing frameworks; in these cases other available tools may show much shorter running times. When used with small data on parallel systems, slower performance is due to the latency of Hadoop's initialization, which is not a problem with big datasets since the time for data processing far exceeds the cost for the start-up latency. Recently, the Hadoop community has started to address this issue and to reduce Hadoop's start-up latency with certain commercial or in memory cluster computing implementations of MapReduce, such as *Apache Spark* (Zaharia *et al*., 2012).

Although inspired by and focused on NGS technology, GMQL can process any genomic region based data in standard data format (e.g. from high-throughput array-based technology such as copy number, SNP, gene, exon, miRNA or protein arrays). Furthermore, GMQL ensures interoperability with both upstream data processing and downstream data analysis tools, by both dealing with data in a variety of standard tab delimited text formats and saving extracted results within tab delimited text files in standard GTF data format; in so doing, GMQL results can then be further processed with usual tools, or visualized in a common genome browser (e.g. the UCSC Genome Browser, Integrated Genome Browser or Integrative Genomics Viewer) for further investigation.

In conclusion, GMQL leverages the many increasingly available and valuable high-level genomic data and their associated biological and clinical metadata for comprehensive and comparative querying for biomedical knowledge discovery, beyond the functionalities offered by existing tools. The provided GMQL toolkit well addresses all the three main challenges in data-intensive genomic analysis: (i) declarativeness: it provides abstractions which allow focusing on the meaning of queries to answer biological questions, rather than on its formulation; (ii) portability: it is portable to various IT infrastructures; (iii) scalability: it scales with data size, providing best performances with large data. The choice of a declarative syntax separated from, but automatically mapped to, the implementation allows for optimizations in the implementation that are transparent to the naive user, enabling researchers to effectively use big genomic data to answer key biological questions.

5 Methods

5.1 Metadata indexing

In order to support efficient search and selection over many data samples based on their possibly very numerous metadata, we use *Apache Lucene* (http://lucene.apache.org/core/), a free/open source information retrieval software library. With Lucene we create and manage indexes over sample metadata and search them to filter the data samples required to answer a query; thus, only data files relevant for the query are loaded in the system and involved in query processing. The created indexing allows any arbitrary Boolean expression over metadata values to be expressed in a query.

5.2 GMQL to Pig Latin translator

Rather than creating a specific execution engine for our GMQL, we developed a GMQL translator to *Pig Latin* (Olston *et al*., 2008), a high-level data-flow language for batch processing of large datasets. This enables us to leverage two of the main features of Pig Latin: automatic execution optimization and extensibility. The former one allows focusing on semantics rather than efficiency of Pig Latin scripts, and the latter one enables the seamless integration of

functions in Java programming language for special-purpose processing; we took advantage of this feature by creating specific genometric Join, Map and Cover functions.

We implemented the GMQL translator in *Racket* (Flatt, 2012), a general-purpose functional programming language in the Lisp/Scheme family that serves as a platform for language creation, design and implementation. The syntax-directed translator has two components, the *lexer* and the *parser*. The former one scans the GMQL query and generates a list of tokens classified as keywords, variable identifiers, numbers, etc; the latter one identifies sub-sequences of the tokens that correspond to grammar rules. Once a rule is matched, a procedure emits the equivalent Pig Latin code. In addition, given that GMQL computes variables with an associated schema, a state describing such schemas is kept up-to-date during the translation.

Datasets of data samples that are referred to in GMQL queries are mapped to suitable Pig Latin variables, where, at execution time, they are loaded using the internal format illustrated in Tables 1 and 2, which allows efficient management and processing in Pig Latin; each dataset that corresponds to a variable is internally mapped into two bags, one for the dataset sample metadata and one for the sample region data. The translation deals with each bag separately; while their correspondence is maintained by the suitable use of sample identifiers. Figure 2 shows the translation of the first GMQL JOIN in the Example 2:

TF1 = JOIN(first(1) after distance 100000, right.distinct)TSS TF;

The function *GenometricPig.Join* computes the join between regions in the *TSS* and *TF* input datasets by capitalizing on the fact that matching regions are at a bound distance, which can be inferred from the query parameters; then, it creates tuples that satisfy the join predicate and loads them in the region and metadata bags of the *TF1* output dataset, where they share the same identifier. Note that a single GMQL operation is translated into many statements of several lines of Pig Latin code; this clearly illustrates the higher level, neatness and ease of GMQL with respect to Pig Latin.

To manage Pig Latin scripts, we use *Apache Pig* (http://pig.apache.org/), an open source platform for analyzing large datasets, which includes a compiler that turns Pig Latin scripts into sequences of MapReduce programs executable on large-scale parallel implementations of the Hadoop open-source framework (http://hadoop.apache.org/). GMQL performance on big datasets has been optimized by executing many operations in parallel; both genomic region data and datasets are partitioned by associating to each partition a distinct data sample and genome chromosome (the id and chr pair), and then computing operations in parallel upon each such partition. Recently,

```
TSS_TF_exp_crs = FOREACH (CROSS TSS_exp_grp, TF_exp_grp)
                 GENERATE ($0,$2), ($1,$3);
TSS_TF_meta_crs = FOREACH (CROSS TSS_meta_grp, TF_meta_grp)
                  GENERATE ($0,$2), ($1,$3);
TSS_TF_meta_flat = FOREACH TSS_TF_meta_crs
                   GENERATE NewId($0, FLATTEN($1));
TF1_meta = UNION (FOREACH TSS_TF_meta_flat
                  GENERATE NewId($0.$0, $0.$1), FLATTEN($1)),
                 (FOREACH TSS_TF_meta_flat
                  GENERATE NewId($0.$0, $0.$1), FLATTEN($2));
DEFINE TF1_joiner = GenometricPig.Join('firstAfterDistance', '100000',
                                       '100000', 'r2_distinct');
TF1_exp = FOREACH (FOREACH TSS_TF_exp_crs
                   GENERATE TF1_joiner($1))
                  GENERATE FLATTEN($0);
```

Fig. 2. Translation of the first GMQL JOIN in Example 2 into Pig Latin

Apache Pig has also added a pluggable execution engine to allow Pig Latin scripts with small input data size running in-process on non-MapReduce frameworks. Although this option offers low performance and is not in line with the GMQL big data focus, we leverage it to enable local GMQL use for users who deal with a few small data samples and do not have access to parallel computing frameworks.

Funding

This work was part of and supported by the 'Data-Driven Genomic Computing [GenData 2020]' PRIN project (2013–2015), funded by the Italian Ministry of the University and Research (MIUR).

Conflict of Interest: none declared.

References

1000 Genomes Project Consortium *et al.* (2010) A map of human genome variation from population-scale sequencing. *Nature*, **467**, 1061–1073.

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

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

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

Flatt,M. (2012) Creating languages in Racket. *Commun. ACM*, **55**, 48–56.

Giardine,B. *et al.* (2005) Galaxy: a platform for interactive large-scale genome analysis. *Genome Res.*, **15**, 1451–1455.

Goecks,J. *et al.* (2010) Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome Biol.*, **11**, R86

Hayden,E.C. (2014) Technology: The $1,000 genome. *Nature*, **507**, 294–295.

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

Li,H. *et al.* (2009) The Sequence Alignment/Map format and SAMtools. *Bioinformatics*, **25**, 2078–2079.

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

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

Nordberg,H. *et al.* (2013) BioPig: a Hadoop-based analytic toolkit for large-scale sequence data. *Bioinformatics*, **29**, 3014–3019.

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

Olston,C. *et al.* (2008) Pig Latin: a not-so-foreign language for data processing. In: L.V.S. Lakshmanan *et al.* (ed) *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*. ACM, New York, 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**, 200–206.

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

Schumacher,A. *et al.* (2014) SeqPig: simple and scalable scripting for large sequencing data sets in Hadoop. *Bioinformatics*, **30**, 119–120.

Sheridan,C. (2014) Illumina claims $1,000 genome win. *Nat. Biotechnol.*, **32**, 115.

Shvachko,K. *et al.* (2010) The Hadoop distributed file system. In: *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, IEEE Computer Society, Washington, DC, pp. 1–10.

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

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

Zaharia,M. *et al*. (2012) Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In: *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation*, USENIX Association, San Jose, CA, pp. 15–28.

Zou,Q. *et al*. (2014) Survey of MapReduce frame operation in bioinformatics. *Brief. Bioinform*., **15**, 637–647.