Specification of GMQL Version 2

Bio-Informatics Group, DEIB, Politecnico di Milano

23/10/2016

Contents

1 Introduction and Motivations

A new technology for reading the DNA, called Next Generation Sequencing (NGS), is changing biological research and will change medical practice, thanks to the low-cost availability of millions of whole genome sequences of a variety of species, and most important of humans. Huge repositories of sequence information are being collected by large consortia of research laboratories by using NGS; among them, ENCODE [12], TCGA [35], the *1000 Genomes Project* [1] and the *100,000 Genomes Project*[1]. These sequences can be assembled with specific experimental data produced at the various research or clinical centers, opening new opportunities for biological discovery and for personalized medicine.

Several organizations are considering genomics at a global level. *Global Alliance for genomics and Health*[2] is a large consortium of over 200 research institutions with the goal of supporting voluntary and secure sharing of genomic and clinical data; their work on data interoperability is producing a data conversion technology[3]. *Google* recently provided an API to store, process, explore, and share DNA sequence reads, alignments, and variant calls, using Google's cloud infrastructure[4]. Parallel frameworks are used to support genomic computing, including *Vertica* (used by Broad Institute and NY Genome Center) and *SciDB* (used by NCBI for storing the data of the 1000 Genome project [1]).

So far, the bio-informatics research community has been mostly challenged by *primary analysis* (production of sequences in the form of shorts DNA segments, or "reads") and *secondary analysis* (alignment of reads to a reference genome and search for specific features, such as variants/mutations and peaks of expression); but the most important emerging problem is the so-called *tertiary analysis*, concerned with multi-sample processing, annotation and filtering of variants, and genome browser-driven exploratory analysis [25]. While secondary analysis targets *raw data* in output from NGS processors by using specialized methods, tertiary analysis targets *processed data* in output from secondary analysis and is responsible of *sense making*, e.g., discovering how heterogeneous regions interact with each other.

The **GenData 2020** research project was conceived to address this challenge, by enabling queries and analysis of processed genomic data[5]. The

[1] http://www.genomicsengland.co.uk/
[2] http://genomicsandhealth.org/
[3] http://ga4gh.org/#/api
[4] https://cloud.google.com/genomics/
[5] http://www.bioinformatics.deib.polimi.it/gendata/

project's main results so far are a *Genomic Data Mode*l (GDM), which encodes processed data in terms of their regions and metadata, and a *Geno-Metric Query Language* (GMQL) for extracting regions of interest from experiments and for computing their properties, with high-level operations for manipulating regions and for measuring their distances [20].

The most relevant and, to the best of our knowledge, original aspects of **GenData 2020** is the targeting of the project towards heterogeneous processed data rather than raw data. World-wide genomic repositories already contain huge amounts of processed data, and actually their value stems from the certification of high-quality processing. While processed data are much smaller than raw data, they can be considered as "big data", because each processed file can contain thousands or even millions of genomic regions. In the **GenData 2020** repository structure, we show that thousands of files can be extracted from the repositories and organized within one dataset, that can be simply referenced by name in GMQL.

Another unique aspect of **GenData 2020** is the inclusion of metadata in the GDM model and of metadata management in the GMQL query language. Each dataset includes in its metadata all known information about each sample (from the sample's preparation up to the patient's phenotype.) GMQL progressively computes both the regions and the metadata of resulting samples. Thus, result samples of every query carry their meta-data, linking them to their contributing input samples, as an indication of **data provenance**; this is very powerful, e.g. for building genotype-phenotype associations.

The **GenData 2020** implementation uses cloud computing. Specifically, Version 1 of the system, developed between 2013 and 2014, translates GMQL to Pig [6] in the context of Hadoop 1 HDFS, while Version 2 of the system, developed in 2015, uses the Spark [7] and Flink [4] frameworks as supported within Hadoop YARN [16]; the need of cloud architectures for genomics is advocated by [30]. The expressive power and flexibility of GenData 2020's data model (GDM) and query language (GMQL) are demonstrated in [20], where we show four very different genomic use cases[6]. GenData 2020 focuses on tertiary data analysis; a similar approach is advocated by *Paradigm4*, a company [7] whose products include genomic adds-on to SciDB [25]. They also concentrate on the *tertiary analysis*, but they advocate the use of specialized databases rather than cloud computing.

[6] Finding ChIP-seq peaks in promoter regions; finding distal bindings in transcription regulatory regions; associating transcriptomics and epigenomics; finding somatic mutations in exons.

[7] Founded by this year's Turing award Mike Stonembraker.

2 Region-based Data Model with Metadata

The Genomic Data Model (GDM) is based on the notions of datasets and samples; datasets are collections of samples, and each sample consists of two parts, the *region data*, which describe portions of the DNA and their features, and the *metadata*, which describe general properties of the sample.

2.1 Motivation

Processed data have a variety of file formats, and typically lack an attribute-based organization. GDM provides a schema to regions, thus it makes data self-describing, as advocated by Jim Gray [15]; however, we don't include data into a database, so as to preserve the possibility for biologists to work with their usual file-based tools.

The schema has a fixed part that guarantees the comparability of regions produced by different kinds of processing, and then a variable part describing the features produced by the various kinds of processing. Although DNA regions are strings of nucleotides[8], we instead associate them with a list of one or more *features*, where each feature is produced by secondary data analysis.

Due to the lack of agreed standards for metadata, we model them as free attribute-value pairs; we expect metadata to include at least the experiment type, the sequencing and analysis method used for data production, the cell line, tissue, experimental condition (e.g., antibody target) and organism sequenced; in case of clinical studies, individual's descriptions including phenotypes. Attributes may have multiple values (e.g., the `Disease` attribute can have both values `'Cancer'` and `'Diabetes'`). Hundreds of datasets and thousands of samples[9] can be queried thanks to the GDM model.

2.2 Definition

A *genomic region* r is a portion of the genome defined by the quadruple of values $< chr, left, right, strand >$, called **region coordinates**, where chr is the chromosome, $left$ and $right$ are the two ends of the region along

[8] DNA can be abstracted as a string of billions of nucleotides (represented by the letters `A,C,G,T`) enclosed within chromosomes (23 in humans), which are disconnected intervals of the string.

[9] We currently store in **GenData 2020** most of ENCODE [12] and TCGA [35] processed data.

the DNA coordinates[10]; *strand* represents the direction of DNA reading[11], encoded as either + or −, and can be missing (encoded as ∗)[12]. Formally, a **sample** s is a triple $< id, R, M >$ where:

- id is the sample identifier of type `long`.
- R is the set of regions of the sample, built as pairs $< c, f >$ of **coordinates** c and **features** f; coordinates are records of four fixed attributes `chr, left, right, strand` which are respectively typed `string, long, long, char`; features are records of typed attributes; we assume attribute names of features to be different, and their types to be any of `char, string, int, long, double, Boolean` (GDM types are available both in Java, Scala, and in the Flink, Spark and Pig frameworks). The `region schema` of s is the list of attribute names used for the identifier, the coordinates and the features.
- M is the set of metadata of the sample, built as attribute-value pairs $< a, v >$, where we assume the type of each value v to be `string`. The same attribute name a can appear in multiple pairs of the same sample (in which case, we say that a is multi-valued).

A **dataset** is a collection of samples with the same region schema and with features having the same types; sample identifiers are unique within each dataset.

2.3 Examples

Each dataset is stored within **GenData 2020** using two tables, one for regions and one for metadata; an example of the two tables for representing a particular experiment, called *ChIP-seq*, is shown in Fig. 1. Note that the region value has an attribute `p_value` of type `real` (representing how significant is the calling of the peak of that genomic region in the ChIP-seq experiment); note also that the `id` attribute is present in both tables;

[10] Species are associated with their *reference genome*; DNA samples are aligned to these references, hence referred to the same system of coordinates; for humans, several references were progressively defined, for instance *hg18*, `hg19` and `hg20/GRCh38`.

[11] DNA is made of two strands rolled-up together in anti-parallel directions, i.e., they are read in opposite directions by the biomolecular machinery of the cell.

[12] According to the University of California at Santa Cruz (UCSC) notation, we use *0-based, half-open inter-base coordinates*, i.e., the considered genomic sequence is $[left, right)$. In this coordinate system, left and right ends can be identical (e.g., when the region represents an insertion in the reference), or consecutive (e.g., when the region represent a single nucleotide polymorphism.

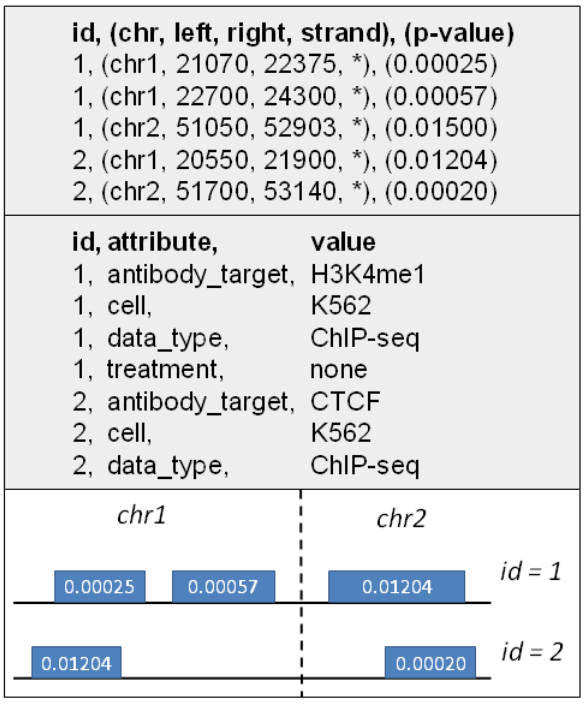

Figure 1: Regions and metadata of a dataset consisting of two samples.

it provides a many-to-many connection between regions and metadata of a sample; e.g., sample 1 has 5 regions and 4 metadata attributes, sample 2 has 4 regions and 3 metadata attributes[13]. The regions of the two samples are within chromosomes 1 and 2 of the DNA, and both are not stranded.

While the above example is simple, GDM supports the schema encoding of any processed data type, e.g., files for mutations, ChIP-seq, DNA-seq, RNA-seq, ChIA-PET, VCF, and SAM/BAM formats. We use GDM also for modeling **annotations**, i.e. regions of the genome with known properties (such as genes, with their exons and introns). Schema encodings and one exemplar instance of DNA-seq and RNA-seq data samples are described in Fig. 2.

[13]Note that the quintuple `(id, chr, left, right, strand)` is not a key of the region table (because a sample can have multiple regions with the same coordinates), and similarly the pair `(id, attribute)` is not a key of the metadata table (because metadata attributes can be multi-valued).

```
DNA-seq (MUTATIONS)
(id, (chr,left,right,strand),
   (A, G, C, T, del, ins, inserted, ambiguous,
    Max, Error, A2T, A2C, A2G, C2A, C2G, C2T))
(1, (chr1, 917179, 917180, *),
   (0, 0, 0, 0, 1, 0, '.', '.',
    0, 0, 0, 0, 0, 0, 0, 0))

RNA-seq (GENE EXPRESSIONS)
(id, (chr,left,right,strand), (source, type, score,
  frame, geneID, transcriptID, RPKM1, RPKM2, iIDR))
(1, (chr8, 101960824, 101964847, *),
 ('GencodeV10', 'transcript', 0.026615, NULL,
   'ENSG00000164924.11', 'ENST00000418997.1',
   0.209968, 0.193078, 0.058))
```

Figure 2: Examples of schema with one instance for two different types of processed data; coordinates and features are enclosed within two records.

3 GenoMetric Query Language

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

```
<variable> = operation(<parameters>) <variables>
```

where each variable stands for a GDM dataset. Operations are either unary (with one input variable), or binary (with two input variables), and construct one result variable.

3.1 General Properties

GMQL operations form a closed algebra: results are expressed as new datasets derived from their operands. All operations produce a result dataset consisting of several samples, whose identifiers are either inherited by the operands or generated by the operation. Each operation separately applies to metadata and to regions; the region-based part of an operation computes the result regions, the metadata part of the operation computes the associated metadata so as to trace the provenance of each resulting sample; identifiers preserve the many-to-many mapping of regions and metadata as discussed in Section 2.3.

Most GMQL operations, although defined upon two connected data structures, are extension of classic relational algebra operations, twisted to the needs of genomics; they are denoted as *relational*. Three domain-specific operations, called `COVER`, (distal) `JOIN` and `MAP`, significantly extend the expressive power of classic relational algebra.

The main design principles of GMQL are *relational completeness* and *orthogonality*. Completeness is guaranteed by the fact that classical algebraic manipulations are all supported, suitably extended and adapted to comply with region-based calculus. Orthogonality is achieved because no operator can be defined as a suitable expression of all other operators; note that the classic abstractions of *grouping* is supported, with the same semantics, in the unary operations `GROUP` and `COVER`, and similarly *joining* is supported, with the same semantics, in the binary operations `JOIN`, `MAP`, `MERGE` and `DIFFERENCE`.

Compared with languages which are currently in use by the bioinformatic community, GMQL is *declarative* (it specifies the structure of the results, leaving its computation to each operation's implementation) and *high-level* (one GMQL query typically substitutes for a long program which embeds calls to region manipulation libraries); the progressive computation of variables resembles other algebraic languages (e.g. *Pig Latin*, [6]). For all these features, GMQL may inspire a change of paradigm in genomics, along a direction that was indicated long ago by Edward T. Codd's seminal papers.

3.2 Predicates Evaluation

Parameters of several operations 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. The region attributes can refer positionally to the schema, i.e., `$0` denotes the first attribute `$1`to the second, and so on. Predicates are either evaluated in the context of regions or of metadata, as follows:

- Predicates on metadata have an existential interpretation over samples: they select the entire sample if it contains some metadata attributes such that the predicate evaluation on their values is true. Formally, for each sample, a simple predicate p expressed as $(A\ comp\ V)$ on metadata M is defined as:

$$p \iff \exists\, (a_i, v_i) \in M : (a_i = A) \wedge (v_i\ comp\ V)$$

When a predicate on metadata uses an attribute which is missing, the predicate is `unknown`; we use three-value (i.e. `true`, `false`, `unknown`) logic for metadata predicates p, and we select samples s for which $p(s)$ is `true` given the above interpretation. The special predicate *missing(A)* is `true` if the attribute A is not present in M.

- Predicates on regions have a classic interpretation: they select the regions where the predicate is true. Legal predicates must use the attributes in the region's schema; when a predicate is illegal, the query is also illegal, and compilation fails.[14] The evaluation of predicates involving two or more regions (essentially join predicates) is defined only when regions have compatible strands; positive and negative strands are incompatible, but they are both compatible with a missing strand.

3.3 Sintactic Conventions

Operations have the general syntax:

```
OUT=OPERATOR(parm-1;..parm-N [; n-parm-1]..[;n-parm-M]) IN-1 [IN-2];
```

Where

- All operations produce an output `OUT`; unari operations apply to a single dataset `(IN-1)`, binary operations apply to two datasets (`IN-1` and `IN-2`).
- `parm` denotes default unnamed parameters for the `OPERATOR`. The semantic of these parameters is inferred from their position.
- `n-parm`: optional parameters that have to be specified in the form of pairs `name: value`. The semantics of each one of these parameters is inferred from its name, therefore their position is irrelevant.

Attributes exist in metadata and regiopns, denoted as follows:

- `<attribute-name>: any-string(.any-string)*` for a generic metadata attribute name.
- `<field-name> : any-string(.any-string)*` for a generic attribute in the region schema.

[14] Region predicates may include metadata attributes, but in such case they are legal iff the metadata attribute is single-valued and not null, and invalid otherwise; in such case, for a given sample, metadata attributes are equivalent to constant values.

The prefix `list` denotes a comma-separated list of elements, e.g. `<list-field-name>` or `<list-attribute-name>`. For what concerns case sensitivity:

- Region and field names are case sensitive: e.g. `pvalue != pValue != PVALUE`
- GMQL keywords are not case sensitive: e.g. `UPSTREAM == upstream ==UpStReAm`

4 Relational GMQL Operations

We next describe relational operations; they include six unary operations (`SELECT`, `PROJECT`, `EXTEND`, `MERGE`, `GROUP` and `SORT`) and two binary operations (`UNION` and `DIFFERENCE`).

4.1 Select

```
<S2> = SELECT([<pm>][;][region: <pr>][;][semijoin: <ps>]) <S1>;
```

where:

- `<pm>`: Expression whose atomic predicates are in the form: attribute-name ($==$ | $>$ | $<$ | $>=$ | $<=$) `(value | decimalNumber)`. Atomic predicates are concatenated by means of the OR, AND and NOT operators; e.g. `antibody==CTCF AND NOT (weight > 100 OR disease == cancer)`.
- `<pr>`: Expression whose atomic predicates are in the form: `field-name` ($==$ | $>$ | $<$ | $>=$ | $<=$) `(value | decimalNumber)`. Atomic predicates are concatenated by means of the OR, AND and NOT operators; e.g. `pvalue < 0.001 OR label==promoter`
- `<ps>`: Semi-join expression in the form: `<list-attribute-name> IN <dataset>`; e.g. `antibody,cell,treatment NOT IN HG_BROAD`

It keeps in the result all the samples which existentially satisfy the predicate on metadata `<pm>` and then selects those regions of selected samples which satisfy the predicate on regions `<pr>`; a sample is legal also when it contains no regions as result of a selection. Identifiers of selected samples of the operand $S1$ are assigned to the result $S2$.

`Semi-join` clauses are used to further select samples; they have the syntax: `<A1>, .., <An> IN <extV>`. Each attribute occurrence `Ai` corresponds to a predicate $p(a_i, a_j)$, where a_i and a_j are attributes with the same

name. a_i belongs to the schema of `A`, a_j to the schema of `extV`. The predicate is true for a given sample s_i of $S1$ with attribute a_i iff there exists a sample in the variable denoted as `extV` with an attribute a_j and the two attributes a_i and a_j share at least one value. Formally, if M_E denotes the metadata of samples of `extV`, then:

$$p(a_i, a_j) \iff \exists\, (a_i, v_i) \in M_i, (a_j, v_j) \in M_E : v_i = v_j$$

A semi-join clause can be constructed as the conjunction of the above simple metadata predicates that refer to the same variable `extV`. Semi-joins are used to connect variables, e.g., in the example below:

```
OUT = SELECT(semijoin: antibody_target IN EXP2) EXP1;
```

samples of `EXP1` are selected only if they have the same `antibody_target` value as in at least one sample of `EXP2`.

4.2 Project

```
<S2> = PROJECT([<Ar1>, .., <Arm>]
       [;][metadata: [<Am1>, .., <Amn>]
       [;][region_update: <Ur1> AS <f1>, .., <Urh> AS <fh>]
       [;][metadata_update: <Um1> AS <h1>, .., <Umk> AS <fk>]) <S1>;
```

It keeps in the result the metadata (`Am`) and region (`Ar`) attributes expressed as parameters[15]. It can also be used to build new attributes as scalar expressions `fi` (e.g., for metadata the `age` from the `birthdate`; for regions, the `length` of a region as the difference between its `right` and `left` ends). If the name of existing schema attributes are used, the operation updates region attributes to new values. Identifiers of the operand $S1$ are assigned to the result `S2`.

4.3 Extend

```
<S2> = EXTEND(<Am1> AS <g1>, .., <Amn> AS <gn>) <S1>;
```

It generates new metadata attributes `Am` as result of aggregate functions `g` applied to region attributes; aggregate functions are applied sample by sample, and resulting tuples are triples with the sample identifier, the attribute

[15] A syntactic variant (using the keywords `ALLBUT`) allows to specify only the attributes that are removed from the result; this variant is very useful with datasets having many region attributes.

name Am, and the computed aggregate value. The supported aggregate functions include COUNT (with no argument), BAG (applicable to attributes of any type) and SUM, AVG, MIN, MAX, MEDIAN, STD (applicable to attributes of numeric types). E.g., in the example below:

```
OUT = EXTEND(RegionCount AS COUNT, MinP AS MIN(Pvalue)) EXP;
```

for each sample of EXP, two new metadata attributes are computed, RegionCount as the number of sample regions, and MinP as the minimum Pvalue of the sample regions.

4.4 Group

```
<S2> = GROUP([<Am1>, .., <Amn>]
      [;][meta_aggregate: <Gm1> AS <g1>, .., <Gmn> AS <gn>]
      [;][region_group: <Ar1>, .., <Arn>]
      [;][region_aggregate: <Gr1> AS <g1>, .., <Grn> AS <gn>]) <S1>;
```

It is used for grouping both regions and metadata according to distinct values of the grouping attributes. For what concerns metadata, each distinct value of the grouping attributes is associated with an output sample, with a new identifier explicitly created for that sample; samples having missing values for any of the grouping attributes are discarded. The metadata of output samples, each corresponding a to given group, are constructed as the union of metadata of all the samples contributing to that group; consequently, metadata include the attributes storing the grouping values, that are common to each sample in the group. New grouping attributes Gm are added to output samples, storing the results of aggregate function evaluations over each group. Examples of typical metadata grouping attributes are the Classification of patients (e.g., as cases or controls) or their Disease values.

When the grouping attribute is multi-valued, samples are partitioned by each subset of their distinct values (e.g., samples with a Disease attribute set both to 'Cancer' and 'Diabetes' are within a group which is distinct from the groups of the samples with only one value, either 'Cancer' or 'Diabetes'). Formally, two samples s_i and s_j belong to the same group, denoted as $s_i \gamma_A s_j$, if and only if they have exactly the same set of values for every grouping attribute A, i.e.

$$s_i \gamma_A s_j \iff \{v | \exists (A, v) \in M_i\} = \{v | \exists (A, v) \in M_j\}$$

Given this definition, grouping has important properties:

- reflexive: $s_i \gamma_A s_i$
- commutative: $s_i \gamma_A s_j \iff s_j \gamma_A s_i$
- transitive: $s_i \gamma_A s_j \wedge s_k \gamma_A s_i \iff s_k \gamma_A s_j$

When grouping applies to regions, by default it includes the grouping attributes `chr, left, right, strand`; this choice corresponds to the biological application of removing *duplicate regions*, i.e. regions with the same coordinates, possibly resulting from other operations, and ensures that the result is a legal GDM instance. Other attributes may be added to grouping attributes (e.g., `RegionType`); aggregate functions can then be applied to each group. The resulting schema includes the attributes used for grouping and possibly new attributes used for the aggregate functions. The following example is used for calculating the minimum `Pvalue` of duplicate regions:

```
OUT = GROUP(Pvalue AS MIN(Pvalue)) EXP;
```

4.5 Merge

```
<S2> = MERGE ([groupby: <AM1>, ..,<AMn>]) <S1>;
```

It builds a dataset consisting of a single sample having as regions all the regions of the input samples (without altering their coordinates, even when overlapping) and as metadata the union of all the attribute-values of the input samples. When a `GROUPBY` clause is present, the samples are partitioned by groups, each with distinct values of grouping metadata attributes (i.e., homonym attributes in the operand schemas) and the merge operation is separately applied to each group, yielding to one sample in the result for each group, as discussed in Section 4.4.

4.6 Order

```
<S2> = ORDER([<Am1> [DESC], .., <Amn> [DESC]]
  [;][meta_top: <k> | [;] meta_topg: <k>]
  [;][region_order: <Ar1> [DESC], .., <Arn> [DESC]]
  [;][region_top: <k> | [;] region_topg: <k>]) <S1>;
```

It orders either samples, or regions, or both of them; order is *ascending* as default, and can be turned to *descending* by an explicit indication. Sorted samples or regions have a new attribute `Order`, added to either metadata, or

regions, or both of them; the value of `Order` reflects the result of the sorting. Identifiers of the samples of the operand $S1$ are assigned to the result $S2$. The clause `TOP <k>` extracts the first k samples or regions, the clause `TOPG <k>` implicitly considers the grouping by identical values of the first $n-1$ ordering attributes and then selects the first k samples or regions of each group. The operation:

```
OUT = ORDER(RegionCount; meta_top: 5;
         region_order: MutationCount DESC; region_top: 7) EXP;
```

extracts the first 5 samples on the basis of their region counter and then, for each of them, 7 regions on the basis of their mutation counter.

4.7 Union

```
<S3> = UNION() <S1> <S2>;
```

It is used to integrate possibly heterogeneous samples of two datasets within a single dataset; each sample of both input datasets contributes to one sample of the result with identical metadata and merged region schema. New identifiers are assigned to each sample.

Two region attributes are considered identical if they have the same name and type; the merging of two schemas is performed by projecting the schema of the second dataset over the schema of the first one. Fields of the first dataset which are missing in the second one are set to `NULL` value, for all the regions of the second operator. For what concerns metadata, attributes are prefixed with the strings `LEFT` or `RIGHT` so as to trace the dataset to which they refer.

4.8 Difference

```
<S3> = DIFFERENCE([joinby: <Att1>, .., <Attn>]) <S1> <S2>;
```

This operation produces a sample in the result for each sample of the first operand $S1$, with identical identifier and metadata. It considers all the regions of the second operand, that we denote as *negative regions*; for each sample $s1$ of $S1$, it includes in the corresponding result sample those regions which do not intersect with any negative region.

When the `JOINBY` clause is present, for each sample $s1$ of the first dataset $S1$ we consider as negative regions only the regions of the samples $s2$ of $S2$ that satisfy the join condition. Syntactically, the clause consists of a list of

attribute names, which are homonyms from the schemas of $S1$ and of $S2$; the strings `LEFT` or `RIGHT` that may be present as prefixes of attribute names as result of binary operators are not considered for detecting homonyms. We formally define a simple equi-join predicate $a_i == a_j$, but the generalization to conjunctions of simple predicates is straightforward. The predicate is true for given samples $s1$ and $s2$ iff the two attributes share at least one value, e.g.:

$$p(a_i, a_j) \iff \exists\, (a_i, v_i) \in M_1, (a_j, v_j) \in M_2 : v_i = v_j$$

The operation:

```
OUT = DIFFERENCE(joinby: antibody_target) EXP1 EXP2;
```

extracts for every pair of samples s_1, s_2 of EXP1 and EXP2 having the same value of `antibody_target` the regions that appear in s_1 but not in s_2; metadata of the result are the same as the metadata of s_1.

5 Domain-Specific Operations

We next focus on *domain-specific* operations, which are more specifically responding to genomic management requirements: the unary operation `COVER` and the binary operations `MAP` and `JOIN`.

5.1 Cover

```
<S2> = COVER/FLAT/SUMMIT/HISTOGRAM (<minAcc>, <maxAcc>
          [; groupby: <Am1>, .., <Amn>]
          [; aggregate: <Ar1> AS <g1>, .., <Arn> AS <gn>]) <S1>;
```

The `COVER` operation responds to the need of computing properties that reflect region's intersections, for example to compute a single sample from several samples which are replicas of the same experiment, or for dealing with overlapping regions (as, by construction, resulting regions are not overlapping.)

Let us initially consider the `COVER` operation with no grouping; in such case, the operation produces a single output sample, and all the metadata attributes of the contributing input samples in $S1$ are assigned to the resulting single sample s in $S2$. Regions of the result sample are built from the regions of samples in $S1$ according to the following condition:

- Each resulting region r in $S2$ is the contiguous intersection of at least `minAcc` and at most `maxAcc` contributing regions r_i in the samples of $S1$ [16]; `minAcc` and `maxAcc` are called **accumulation indexes**[17].

Resulting regions may have new attributes Ar, calculated by means of aggregate expressions over the attributes of the contributing regions. `Jaccard Indexes`[18] are standard measures of similarity of the contributing regions r_i, added as default region attributes. When a `GROUPBY` clause is present, the samples are partitioned by groups, each with distinct values of grouping metadata attributes (i.e., homonym attributes in the operand schemas) and the cover operation is separately applied to each group, yielding to one sample in the result for each group, as discussed in Section 4.4.
For what concerns variants:

- `FLAT` returns the union of all the regions which contribute to the `COVER` (more precisely, it returns the contiguous region that starts from the first end and stops at the last end of the regions which would contribute to each region of the `COVER`).

- `SUMMIT` returns only those portions of the result regions of the `COVER` where the maximum number of regions intersect (more precisely, it returns regions that start from a position where the number of intersecting regions is not increasing afterwards and stops at a position where either the number of intersecting regions decreases, or it violates the max accumulation index).

- `HISTOGRAM` returns the nonoverlapping regions contributing to the cover, each with its accumulation index value, which is assigned to the **AccIndex** region attribute.

Example. Fig. 3 shows three applications of the `COVER` operation on three samples, represented on a small portion of the genome; the figure shows the

[16] When regions are stranded, cover is separately applied to positive and negative strands; in such case, unstranded regions are accounted both as positive and negative.

[17] The keyword `ANY` can be used as `maxAcc`, and in this case no maximum is set (it is equivalent to omitting the `maxAcc` option); the keyword `ALL` stands for the number of samples in the operand, and can be used both for `minAcc` and `maxAcc`. Cases when `maxAcc` is greater than `ALL` are relevant when the input samples include overlapping regions.

[18] The `JaccardIntersect` index is calculated as the ratio between the lengths of the intersection and of the union of the contributing regions; the `JaccardResult` index is calculated as the ratio between the lengths of the result and of the union of the contributing regions.

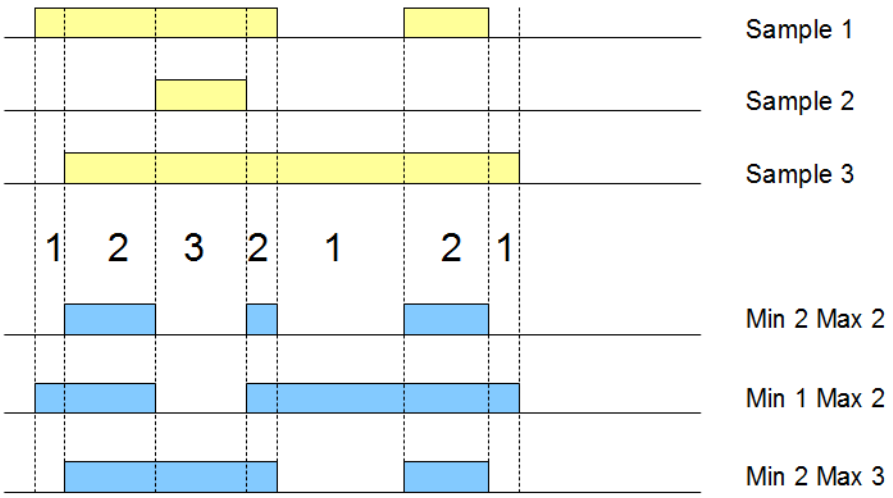

Figure 3: Accumulation index and `COVER` results with three different `minAcc` and `maxAcc` values.

values of the accumulation index and then the regions resulting from setting the `minAcc` and `maxAcc` parameters respectively to $(2, 2)$, $(1, 2)$, and $(2, 3)$.

The following `COVER` operation produces output regions where at least 2 and at most 3 regions of `EXP` overlap, having as resulting region attributes the min `pValue` of the overlapping regions and their Jaccard indexes; the result has one sample for each input `cell`.

```
RES = COVER(2, 3; groupby: cell; aggregate:
                                    pValue AS MIN(pValue)) EXP;
```

5.2 Map

```
<S3> = MAP([<Ar1> AS <g1>, .., <Arn> AS <gn>]
       [;][joinby: <Am1>, .., <Amn>]) <S1> <S2>;
```

`MAP` is a binary operation over two datasets, respectively called **reference** and **experiment**. Let us consider one reference sample, with a set of reference regions; the operation computes, for each sample in the experiment, aggregates over the values of the experiment regions that intersect with each reference region; we say that *experiment regions are mapped to reference regions*. The operation produces a matrix structure, called **genomic space**,

where each experiment sample is associated with a row, each reference region with a column, and each matrix row is a vector of numbers[19]. Thus, a `MAP` operation allows a quantitative reading of experiments with respect to the reference regions; when the biological function of the reference regions is not known, the `MAP` helps in extracting the most interesting regions out of many candidates.

We first consider the basic `MAP` operation, without `JOINBY` clause. For a given reference sample s_1, let R_1 be the set of its regions; for each sample s_2 of the second operand, with $s_2 =< id_2, R_2, M_2 >$ (according to the GDM notation), the new sample $s_3 =< id_3, R_3, M_3 >$ is constructed; id_3 is generated from id_1 and id_2[20], the metadata M_3 are obtained by merging metadata M_1 and M_2, and the regions $R_3 = \{< c_3, f_3 >\}$ are created such that, for each region $r_1 \in R_1$, there is exactly one region $r_3 \in R_3$, having the same coordinates (i.e., $c_3 = c_1$) and having as features f_3 obtained as the concatenation of the features f_1 and the new attributes computed by the aggregate functions g specified in the operation; such aggregate functions are applied to the attributes of all the regions $r_2 \in R_2$ having a non-empty intersection with r_1. A default aggregate `Count` counts the number of regions $r_2 \in R_2$ having a non-empty intersection with r_1. For each region, a field named `count_LeftDSName_RighDSName` is added, storing the result of `Count` aggregate. The operation is iterated for each reference sample, and generates a sample-specific genomic space at each iteration.

When the `JOINBY` clause is present, for each sample $s1$ of the first dataset $S1$ we consider the regions of the samples $s2$ of $S2$ that satisfy the join condition. Syntactically, the clause consists of a list of attribute names, which are homonyms from the schemas of S1 and of S2; the strings `LEFT` or `RIGHT` that may be present as prefixes of attribute names as result of binary operators are not considered for detecting homonyms.

Example. Fig. 4 shows the effect of this `MAP` operation on a small portion of the genome; the input consists of one reference sample with 3 regions and three mutation experiment samples, the output consists of three samples, each with the same regions as the reference sample, whose features corresponds to the number of mutations which intersect with those regions. The result can be interpreted as a (3×3) genome space.

[19] Biologists typically consider the transposed matrix, because there are fewer experiments (on columns) than regions (on rows). Such matrix can be observed using heat maps, and its rows and/or columns can be clustered to show patterns.

[20] The implementation generates identifiers for the result by applying hash functions to the identifiers of operands, so that resulting identifiers are unique; they are identical if generated multiple times for the same input samples.

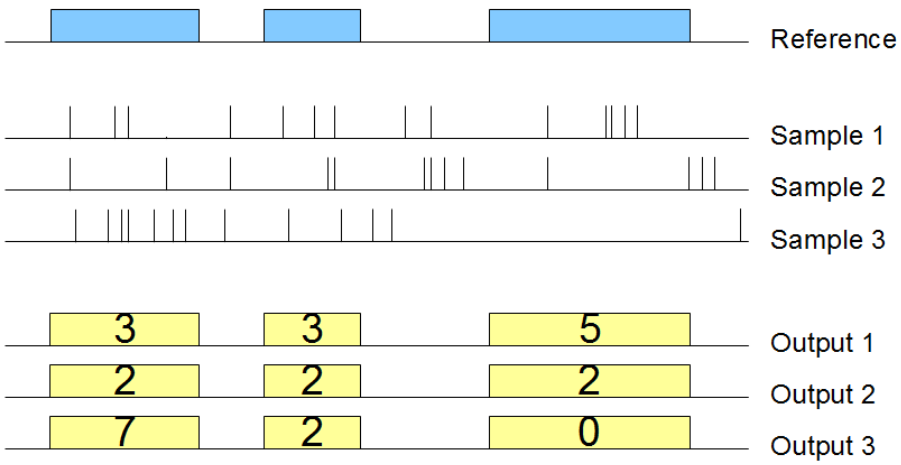

Figure 4: Example of map using one sample as reference and three samples as experiment, using the Count aggregate function.

In the example below, the `MAP` operation counts how many mutations occur in known genes, where the dataset `EXP` contains DNA mutation regions and `GENES` contains the genes.

```
RES = MAP() GENES EXP;
```

5.3 Join

```
<S3> = JOIN([<genometric-pred>][;] [output: <coord-gen>]
        [;] [joinby: <Am1>, .., <Amn>]) <S1> <S2>;
```

The `JOIN` operation applies to two datasets, respectively called **anchor** (the first one) and **experiment** (the second one), and acts in two phases (each of them can be missing). In the first phase, pairs of samples which satisfy the `joinby` predicate (also called meta-join predicate) are identified; in the second phase, regions that satisfy the **genometric predicate** are selected. The meta-join predicate allows selecting sample pairs with appropriate biological conditions (e.g., regarding the same cell line or antibody); syntactically, it is expressed as a list of homonym attributes from the schemes of `S1` and `S2`, as previously. The genometric join predicate allows expressing a variety of distal conditions, needed by biologists. The anchor is used as startpoint in evaluating genometric predicates (which are not symmetric). The join result is constructed as follows:

- The meta-join predicates initially selects pairs s_1 of $S1$ and s_2 of $S2$ that satisfy the joinby condition. If the clause is omitted, then the Cartesian product of all pairs s_1 of $S1$ and s_2 of $S2$ are selected. For each such pair, a new sample s_{12} is generated in the result, having an identifier id_{12}, generated from id_1 and id_2, and metadata given by the union of metadata of s_1 and s_2.

- Then, the genometric predicate is tested for all the pairs $< r_i, r_j >$ of regions, with $r_1 \in s_1$ and $r_j \in s_2$, by assigning the role of **anchor region**, in turn, to all the regions of $s1$, and then evaluating the genometric predicate condition with all the regions of $s2$. From every pair $< r_i, r_j >$ that satisfies the join condition, a new region is generated in s_{12}.

From this description, it follows that the join operation yields results that can grow quadratically both in the number of samples and of regions; hence, it is the most critical GMQL operation from a computational point of view.

Genometric predicates are based on the **genomic distance**, defined as the number of bases (i.e., nucleotides) between the closest opposite ends of two regions, measured from the right end of the region with left end lower coordinate.[21] A genometric predicate is a sequence of distal conditions, defined as follows:

- `UP/DOWN`[22] denotes the *upstream* and *downstream* directions of the genome. They are interpreted as predicates that must hold on the region s_2 of the experiment; `UP` is true when s_2 is in the *upstream genome* of the anchor region[23]. When this clause is not present, distal conditions apply to both the directions of the genome.

- `MD(K)`[24] denotes the *minimum distance* clause; it selects the K regions of the experiment at minimal distance from the anchor region. When

[21] Note that with our choice of interbase coordinates, intersecting regions have distance less than 0 and adjacent regions have distance equal to 0; if two regions belong to different chromosomes, their distance is undefined (and predicates based on distance fail).

[22] Also: UPSTREAM, DOWNSTREAM.

[23] *Upstream* and *downstream* are technical terms in genomics, and they are applied to regions on the basis of their *strand*. For regions of the *positive strand* (or for *unstranded regions*), `UP` is true for those regions of the experiment whose right end is lower than the left end of the anchor, and `DOWN` is true for those regions of the experiment whose left end is higher than the right end of the anchor. (Remaining regions of the experiment are overlapping with the anchor region.) For the *negative strand*, ends and disequations are exchanged.

[24] Also: MINDIST, MINDISTANCE.

there are ties (i.e., regions at the same distance from the anchor region), regions of the experiment are kept in the result even if they exceed the K limit.

- `DLE(N)`[25] denotes the *less-equal distance* clause; it selects all the regions of the experiment such that their distance from the anchor region is less than or equal to N bases[26].

- `DGE(N)`[27] denotes the *greater-equal distance* clause; it selects all the regions of the experiment such that their distance from the anchor region is greater than or equal to N bases.

Genometric clauses are composed by strings of distal conditions; we say that a genometric clause is **well-formed** iff it includes the *less-equal distance* clause; we expect all clauses to be well formed, possibly because the clause `DLE(Max)` is automatically added at the end of the string, where `Max` is a problem-specific maximum distance.
Example. The following strings are legal genometric predicates:

```
DGE(500), UP, DLE(1000), MD(1)
DGE(50000), UP, DLE(100000), (S1.left - S2.left > 600)
DLE(2000), MD(1), DOWN
MD(100), DLE(3000)
```

Note that different orderings of the same distal clauses may produce different results; this aspect has been designed in order to provide all the required biological meanings.
Examples. In Fig. 5 we show an evaluation of the following two clauses relative to an anchor region: `A: MD(1), DGE(100)`; `B: DGE(100), MD(1)`. In case A, the `MD(1)` clause is computed first, producing one region which is next excluded by computing the `DGE(100)` clause; therefore, no region is produced. In case B, the `DGE(100)` clause is computed first, producing two regions, and then the `MD(1)` clause is computed, producing as result one region[28].

[25] Also: DIST <= N, DISTANCE <= N.

[26] DLE(-1) is true when the region of the experiment overlaps with the anchor region; DLE(0) is true when the region of the experiment is adjacent to or overlapping with the anchor region.

[27] Also: DIST >= N, DISTANCE >= N.

[28] The two queries can be expressed as: *produce the minimum distance region iff its distance is less than 100 bases* and *produce the minimum distance region after 100 bases.*

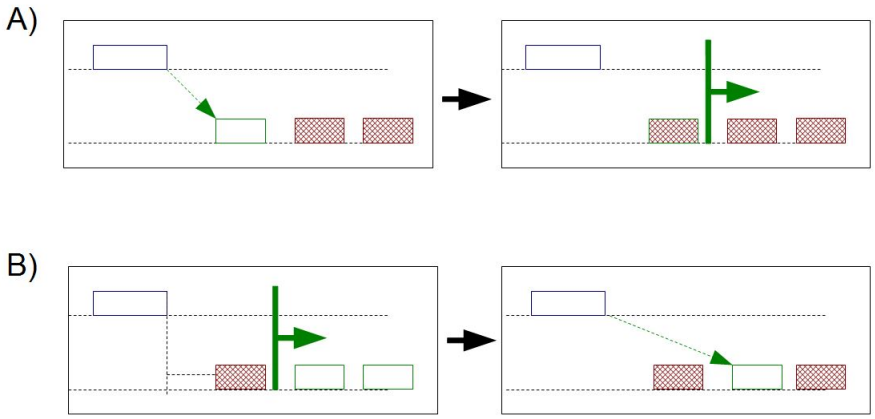

Figure 5: Different semantics of genometric clauses due to the ordering of distal conditions; excluded regions are gray.

Similarly, the clauses `A: MD(1), UP` and `B: UP, MD(1)` may produce different results, as in case A the minimum distance region is selected regardless of streams and then retained iff it belongs to the upstream of the anchor, while in case B only upstream regions are considered, and the one at minimum distance is selected.

Next, we discuss the structure of resulting samples. Assume that regions r_i of s_i and r_j of s_j satisfy the genometric predicate, then a new region r_{ij} is created, having merged features obtained by concatenating the feature attributes of the first dataset with the feature attributes of the second dataset as discussed in Section 4.7. The coordinates c_{ij} are generated according to the `coord-gen` clause, which has four options [29]:

1. `LEFT` assigns to r_{ij} the coordinates c_i of the anchor region.
2. `RIGHT` assigns to r_{ij} the coordinates c_j of the experiment region.
3. `INT` assigns to r_{ij} the coordinates of the intersection of r_i and r_j; if the intersection is empty then no region is produced.
4. `CAT` (also: `CONTIG`) assigns to r_{ij} the coordinates of the concatenation of r_i and r_j (i.e., the region from the lower left end between those of r_i and r_j to the upper right end between those of r_i and r_j).

[29] If the operation applies to regions with the same strand, the result is also stranded in the same way; if it applies to regions with different strands, the result is not stranded.

Example. The following join searches for those regions of particular ChIP-seq experiments, called histone modifications (HM), that are at a minimal distance from the transcription start sites of genes (TSS), provided that such distance is greater than 120K bases[30]. Note that the result uses the coordinates of the experiment.

```
RES = JOIN(MD(1), DGE(120000); output: RIGHT) TSS HM;
```

6 Utility Operations

6.1 Materialize

```
MATERIALIZE <S1> INTO file_name;
```

The `MATERIALIZE` operation saves the content of a dataset `S1` in a file, whose name is specified, and registers the saved dataset in the system to make it seamlessly usable in other GMQL queries. All datasets defined in a GMQL query are, by default, temporary; to see and preserve the content of any dataset generated during a GMQL query, the dataset must be materialized. Any dataset can be materialized, however the operation is time expensive; for best performance, materialize the relevant data only.

7 Full Biological Example

This example uses a `MAP` operation to count the peak regions in each ENCODE ChIP-seq sample that intersect with a gene promoter (i.e., regulatory region); then, in each sample it projects over the promoters with at least one intersecting peak and counts these promoters. Finally, it extracts the top-3 samples with the highest number of such promoters.

```
HM_TF = SELECT(dataType == 'ChipSeq') ENCODE;
PROM = SELECT(annotation == 'promoter') ANNOTATION;
PROM1 = MAP() PROM HM_TF;
PROM2 = PROJECT(region: count_PROM_HM_TF >= 1) PROM1;
PROM3 = EXTEND(prom_count AS COUNT()) PROM2;
PROM_res = ORDER(DESC prom_count; meta_top: 3) PROM3;
MATERIALIZE PROM_res INTO res;
```

[30] This query is used in the search of *enhancers*, i.e., parts of the genome which have an important role in gene activation.

The query was executed over 2,423 samples including a total of 83,899,526 peaks, which were mapped to 131,780 promoters, producing as result 29 GB of data; next, promoters with peaks were counted, and the 3 samples with more of such promoters were selected, having between 30K and 32K promoters each.

ID	ATTRIBUTE	VALUE
131	order	1
131	antibody	RBBP5
131	cell	H1-hESC
131	count	32028
133	order	2
133	antibody	SIRT6
133	cell	H1-hESC
133	count	30945
113	order	3
113	antibody	H2AFZ
113	cell	H1-hESC
113	count	30825

Figure 6: Metadata excerpt of resulting samples

The `PROM_res` result variable includes both regions and metadata; the former indicate the top 3 interesting promoter regions (that can be inspected using viewers, e.g., genome browsers), the latter allow tracing provenance of resulting samples and associating the extracted genomic information with the phenotypes. Fig. 6 shows 4 metadata attributes of the resulting samples: the `order` of the sample, the `antibody` and `cell` type (normal embryonic stem cells) used in the experiment preparation, and the promoter region `count`.

References

[1] The 1000 Genomes Consortium, An integrated map of genetic variation from 1,092 human genomes. *Nature*, 491, 56-65, November 2012.

[2] F. Afrati et al., Bounds for Overlapping Interval Join of Map Reduce. *Workshop Proceedings, EDBT/ICDT*, 2015.

[3] A. Alexandrov et al. The Strathosphere platform for big data analytics. *VLDB Journal* 23(6), 939-964, 2014.

[4] Apache Flink. http://flink.apache.org/

[5] Apache Lucene. http://lucene.apache.org/core/

[6] Apache Pig. http://pig.apache.org/

[7] Apache Spark. http://spark.apache.org/

[8] V. Bafna et al. Abstractions for genomics. *Commun. ACM*, 56(1):83-93, 2013.

[9] M. Cereda et al. GeCo++: a C++ library for genomic features computation and annotation in the presence of variants. *Bioinformatics*, 27(9):1313-1315, 2011.

[10] J. Ekanayake et al. MapReduce for data intensive scientific analyses. In *Proc. IEEE eScience*, 277-284, 2008.

[11] S. Ewen et al., Spinning fast iterative data flows. *PVLDB 2012*, 1268-1279.

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

[13] Galaxy. http://galaxyproject.org/

[14] H. Gunadhi and A. Segev. Query processing algorithms for temporal intersection joins. In *Proc. IEEE ICDE*, 336-344, 1991.

[15] T. Hey at al. Jim Gray on eScience: a Transformed Scientific Method, In *The fourth paradigm. Data-intensive scientific discovery*, Microsoft Research, Redmond, WA, XVII-XXXI, 2009.

[16] Hadoop 2. http://hadoop.apache.org/docs/stable/

[17] F. Hueske et al. Opening the black boxes in dataflow optimization. *PVLDB 2012*, 1256-1267.

[18] W.J. Kent, The human genome browser at UCSC. *Genome Res.*, 2002; 12(6):996-1006.

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

[20] M. Masseroli, P. Pinoli, F. Venco, A. Kaitoua, V. Jalili, F. Palluzzi, H. Muller, S. Ceri. GenoMetric Query Language: A novel approach to large-scale genomic data management. *Bioinformatics*, 2015; 31(12):1881-1888.

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

[22] H. Nordberg et al. BioPig: a Hadoop-based analytic toolkit for large-scale sequence data. *Bioinformatics*, 29(23):3014-3019, 2013.

[23] C. Olston et al. Pig Latin: A not-so-foreign language for data processing. *ACM-SIGMOD*, 1099-1110, 2008.

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

[25] Anonymous paper, Accelerating Bioinformatics Research with New Software for Big Data to Knowledge (BD2K), Paradigm4 Inc., Waltham, MA, 1-16, 2015 (downloaded from: http://www.paradigm4.com/ on June 2015.)

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

[27] U. Röhm and J. Blakeley. Data management for high-throughput genomics. In *Proc. CDIR*, 1-10, 2009.

[28] A. Schumacher et al. SeqPig: simple and scalable scripting for large sequencing data sets in Hadoop. *Bioinformatics*, 30(1):119-120, 2014.

[29] K. Shvachko et al. The Hadoop distributed file system. In *Proc. MSST*, 1-10, 2010.

[30] L.D. Stein. The case for cloud computing in genome informatics. *Genome Biol.*, 11(5):207, 2010.

[31] S. Tata et al. Declarative Querying for Biological Sequences. In *Proc. IEEE ICDE*, 87:99, 2006.

[32] S. Tata et al. Periscope/SQL: Interactive exploration of biological sequence databases. In *Proc. VLDB*, 1406-1409, 2007.

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

[34] R. Xin et al. Shark: SQL and Rich Analytics at Scale. In *Proc. ACM-SIGMOD*, June 2013.

[35] J. N. Weinstein et al. The Cancer Genome Atlas Pan-Cancer analysis project. *Nat Genet.*, 45(10):1113-1120, 2013.

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

[37] M. Zaharia et al. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proc. USENIX*, 15-28, 2012.

[38] M. Zaharia et al. Discretized Streams: Fault-Tolerant Streaming Computation at Scale. In *Proc.M SOSP*, November 2013.