Optimal Binning for Genomics

Andrea Gulino, Abdulrahman Kaitoua and Stefano Ceri

Abstract—Genome sequencing is expected to be the most prolific source of big data in the next decade; millions of whole genome datasets will open new opportunities for biological research and personalized medicine. Genome sequences are abstracted in the form of interesting regions, describing abnormalities of the genome. The parallel execution on the cloud of complex operations for joining and mapping billions of genomic regions is increasingly important. Genome binning, i.e. partitioning of the genome into small-size segments, adapts classic data partitioning methods to genomics; region distributions to bins must reflect operation-specific correctness rules. As a consequence, determining the optimal bin size for such operations is a complex mathematical problem, whose solution requires careful modeling. The main result of this paper is the mathematical formulation and solution of the optimal binning problem for join and map operations in the context of GMQL, a query language over genomic regions; the model is validated by experiments showing its accuracy and sensitivity to the variation of operations' parameters. We also optimize sequences of operations by inheriting the binning between two consecutive operations and we show the deployment of GMQL and the tuning of the proposed model on different cloud computing systems.

Index Terms—Big data applications, Query processing, Genomics, Partitioning algorithms, Optimization.

1 Introduction

Next Generation Sequencing (NGS), a high-throughput, massively parallel technology for reading the DNA, has made gigantic steps in the last 10 years. The cost of producing a complete human sequence dropped to 1000 US$ in 2015 [30] and is expected to drop below 100 US$ in the next 2-3 years. Each sequence produces raw data in the form of short reads of genome strings, whose storage requires about 200 GByte; between 100 million and 2 billion human genomes will be sequenced by 2025 [39], thereby generating the biggest big data problem for the mankind.

Technological development brings about new methods for extracting signals from the genome, and this in turn helps us in better understanding the genome. Our concept of genome has evolved, from a long string of 3,2 billions of base pairs to a living system producing signals, to be integrated and interpreted. Signals include variants (positions or regions of the genome where the code of an individual differs from the reference genome), gene expression (measuring gene activity in protein transcription) and peaks of expression (positions of the genome with an increase of short reads which indicate specific biological events, such as the binding of a protein to the DNA).

Signals are produced as result of long and complex bioinformatics pipelines. In particular, analysis of NGS data is classified as primary, secondary and tertiary[1]. Primary analysis is essentially responsible of producing raw data. Secondary analysis is responsible of producing the so-called *processed data*, by extracting (calling) the signal from raw data and aligning them to the reference genome. Thousands of processed datasets are becoming available at large sequencing centers, are being assembled by large international consortia and made available for secondary research use; they include the Enciclopedia of DNA Elements (ENCODE) [20], The Cancer Genome Atlas (TCGA) [45], and the 1000 Genomes Project [1].

Tertiary analysis is responsible of exploring, querying and integrating processed data, so as to give answers to complex biological and clinical questions, ultimately yielding to personalized medicine. So far, very few systems are specifically dedicated to tertiary data analysis. Among them, SciDB, a scientific database produced by the spinoff company Paradigm4 [7] and DeepBlue, produced by the BluePrint consortium [5].

We are currently developing a new, holistic approach for tertiary data analysis. Our approach is based on a new, high-level query language, called GenoMetric Query Language (GMQL) [28], which enables building new datasets from a repository of existing datasets, using algebraic operations. Our approach is truly multidisciplinary, as it combines data modeling and management, big data, cloud computing, systems architecture and parallel algorithms. The current implementation of GMQL is available since March 2016 at CINECA supercomputing site[2].

In a previous paper [23] we presented our framework for the parallel implementation of GMQL operations on the cloud and the algorithms for the implementation of domain-specific GMQL operations using Flink [8] and Spark [9]. These algorithms use *binning*, i.e. a partitioning of the genome into portions of equal size so as to enable parallelism, but we did not address the problem of binning optimization. GMQL has been so far using a single bin size for each operation, set at the beginning of query execution.

In this paper, we solve the problem of determining the optimal binning of JOIN and MAP, the most important domain-specific GMQL operations: together with SELECTION and PROJECTION, they allow defining a particular

• *A. Gulino and S. Ceri are at the Dipartimento di Elettronica ed Informazione e Bioingegneria, Politecnico di Milano, Italy; A. Kaitoua is at DFKI Berlin. E-mail: {firstname.lastname}@polimi.it*

1. http://blog.goldenhelix.com/grudy/a-hitchhiker%E2%80%99s-guide-to-next-generation-sequencing-part-2

2. GMQL-V2, http://www.gmql.eu/interfaces/

class of GMQL programs, denoted as *conjunctive GMQL programs*, which constitute the core of the language and are used by most applications. We present analytical models for determining the optimal bin size of each operation independently, and then an approach to the optimization of consecutive operations within conjunctive GMQL queries. In order to predict the optimal models, we also introduce genomic profiling, which takes into account the specificity of genomic datasets. The combination of profiling and bin optimization, included within the latest release of GMQL, provides a comprehensive approach to query optimization for tertiary data management in genomics, whose applicability goes beyond GMQL. We show some examples where a small underestimation or overestimation of the bin size doubles the execution time.

The remaining of this paper is structured as follows. In order to make this paper self-contained, Section 2 summarizes the GMQL data model, the JOIN and MAP operations, and the strategy for binning the genome as a function of the specific JOIN and MAP conditions. Together with the data model and operations we also introduce the definition and evaluation of their profiles, while the detailed implementation of JOIN and MAP operations using Flink and Spark is omitted, and can be found on [23]. Section 3 then presents the analytical models for computing the optimal bin size of JOIN and MAP, and Section 4 provides the validation of the models both on synthetic and real datasets, with a sensitivity analysis showing the dependency of the optimal choice of binning from the query parameters. Section 5 describes the optimization of sequences of operations, achieved by reusing bin assignments; Section 6 describes the deployment of GMQL over clouds. Finally, we present the related work and conclusions.

2 Genomic Data Model and Operations

In this section, we summarize the GMQL data model and the JOIN and MAP operations, which are the essential ingredients of conjunctive GMQL queries.

2.1 Region-Based Data Model

In this paper, genomic information is organized within *DNA regions*, i.e. portions of the genome placed within one chromosome and further characterized by a start and stop position, and possibly by a strand, i.e. the direction of reading of the DNA. Regions are aligned with respect to a reference genome, therefore regions are related to each other by a single system of coordinates. Every region carries a *value* which describes the signal associated with the region - hence, it can describe mutations, gene expression, peaks of expressions, or other interesting properties of the genome. In general, the region value is complex and is associated with a given signature.

The regions which are produced by a given experimental condition (e.g. a specific DNA treatment for a given cell) are contained within a file or *sample*. We then collect into collections, called *datasets*, samples that are produced by the use of the same technology - hence, their value has the same signature. Datasets are named and each sample within a dataset has a unique identifier. Each dataset has a *schema*, which is a list of typed attributes. The first five attributes, describing the sample's identity and region's coordinates, are fixed (`Sample-ID`, `Chromosome`, `Start`, `Stop`, `Strand`) while the subsequent attributes describe the region's value and depend on the specific technology used for the experiment [3].

TABLE 1: Parameters of dataset profiles

Feature	Description
S	Number of samples in a dataset
L	Size of the region schema
N_i	Number of regions in sample i
w	Average region length
u	Useful space

2.1.1 Profiles

Profiles are used in order to quantitatively define the properties that better characterize a genomic dataset from the point of view of query optimization, whose main aspect is the determination of the optimal bin size. Table 1 lists the main profile parameters. They include some typical counters, such as the number of samples S, the size of the region's schema L, and the number of regions in each sample N_i. Therefore, the initial size of each dataset is given by:

$$size(DS) = \sum_i (L \times N_i)$$

Specific parameters which are needed in a region-based calculus are: w (average region length) which is essential for computing the selectivity of join predicates, and u (useful space), that measures the amount of space in a sample that is actually occupied by some regions, since the initial and final parts of chromosomes are typically empty. By assuming that, in a sample, regions within each chromosome are uniformly distributed along a single interval of coordinates, the useful space u is computed as:

$$u = \sum_{c \in C} M_c - m_c$$

where C is the set of distinct chromosomes appearing in the sample, M_c and m_c are respectively the maximum and minimum coordinates of regions belonging to chromosome c. We opted for a single parameter per sample instead of chromosome-specific parameters so as to keep profiling more manageable.

We assume that profiles are available for each initial dataset. We estimate the profiles of intermediate result produced by a query by applying heuristics to its input profiles. Each heuristic models the transformations performed by an operator of the language to its input datasets. In one case of join, since no heuristic is available, we must dynamically reprofile the result (see Section 2.2.2).

2.2 Conjunctive GMQL Programs

A GMQL query is a sequence of GMQL operations with the following structure:

3. The GMQL data model includes also metadata attributes as a fundamental concept, but they are not required for explaining optimal binning, hence we omit their description; see [28].

```
<var> = operation(<parameters>) <vars>
```

where each variable stands for a GMQL dataset. Operations are either unary (with one input variable) or binary (with two input variables) ad produce a result variable. The result variable corresponds to a new dataset consisting of several samples, whose content is computed by the operation; in particular, sample identifiers are either inherited by the operands or generated by the operation.

Most GMQL operations are extensions of classic relational algebra operations, twisted to the needs of genomics. In [28], we demonstrated the expressive power and flexibility of GMQL through multiple biological examples, including finding distal bindings in transcription regulatory regions, associating transcriptomics and epigenomics, and finding somatic mutations in exons. Compared with languages which are currently in use by the bioinformatics 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 embedded within a programming language such as R or Python).

The core of GMQL is constituted by the classical operations of SELECT, PROJECT and JOIN, extended by the MAP operation - used to generate query results which are suitable for further data analysis. SELECT and PROJECT are rather standard[4]. Hence, we briefly summarize only JOIN and MAP; we also describe how the profile of their results can be computed.

2.2.1 Join

The `JOIN` operation applies to two datasets, respectively called **anchor** and **experiment**, and produces a result sample for every pair of samples of the operand datasets, whose identifier is obtained by applying a hash function to the identifiers of the operand samples. The coordinates of resulting regions are computed according to four region composition options: LEFT and RIGHT project resulting regions over the left or right operand dataset, INT produces their intersection and CONCAT produces their concatenation. Thus, the join operation produces 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.

The regions within each result sample are generated from the regions of the operand samples that satisfy a *genometric predicate*, based on the notion of **genomic distance**. Distance is defined as the number of bases between the closest opposite ends of two regions. Intersecting regions have distance less than 0, adjacent regions have distance equal to 0. If two regions belong to different chromosomes, their distance is undefined (and predicates based on distance fail).

A genometric predicate is a sequence of distal conditions, defined as follows:

- `UP/DOWN` denotes the *upstream* and *downstream* directions of the genome. They are interpreted as predicates that must hold on the region of the experiment; `UP` is true when it is in the *upstream genome* of the anchor region[5]. When this clause is not present, distal conditions apply to both the directions of the genome.
- `MD(K)` denotes the *minimum distance* clause; it selects the K regions of the experiment at minimal distance from the anchor region. When 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)` denotes the *less 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.
- `DGE(N)` denotes the *greater 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. They define a *search space* for each anchor region, i.e. intervals of coordinates in which the closest ends of an experiment region must fall in order to satisfy the predicate (see Section 3.1).

A genometric predicate is **well-formed** only if it includes the *less 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. As discussed in [23], the `DLE` and `UP/DOWN` clauses are commutative with the other clauses and can be pushed at the beginning of the string, thus delimiting the search space. After clause rewriting, the clauses appearing in the predicate before the `MD` clause - if present - are denoted as *first-step clauses*; they include a `DLE` clause and may include a `DGE` and a `UP/DOWN` clause.

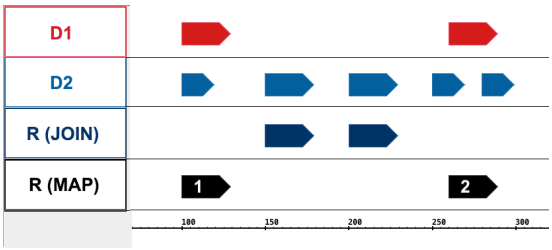

Fig. 1: Example of Map and Join input and output datasets visualized through the Integrated Genome Broswer (http://bioviz.org). Genomic regions are all in the positive strand.

Example 2.1. Consider the following query:

```
A = SELECT() D1;
E = SELECT() D2;
R = JOIN(DLE(40),DGE(15); output:RIGHT) A E;
```

computed on the following anchor and experiment datasets:

4. SELECT and PROJECT enable selecting samples and regions, or projecting parts of the region value; their profile evaluation requires to compute S, N_i and L respectively. The first selection causes also the loading of samples, thus S and N_i are precisely computed from sample-specific parameters at the end of loading.

5. *Upstream* and *downstream* are genomic predicates. For regions of the *positive strand*, `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. For the *negative strand*, ends and inequalities are exchanged.

```
D1 : chromosome, start, stop, strand
chr1 100 130 +
chr1 260 290 +

D2: chromosome, start, stop, strand
chr1 100 120 +
chr1 150 180 +
chr1 200 230 +
chr1 250 270 +
chr1 280 300 +
```

The result, shown in the third track of Fig. 1, is made of those experiment regions whose genomic distance from an anchor region was between 15 and 40 bases:

```
R: chromosome, start, stop, strand
chr1 150 180 +
chr1 200 230 +
```

2.2.2 Profiles Estimation for Join Results

Profiles of intermediate result are computed by applying heuristics to the input profiles; each heuristic models the transformations performed by an operator of the language to the features of its input datasets[6].

The output profile of the result of a join operation is estimated from the profiles of its input variables with the heuristics presented in the following. The notation $i \bowtie j$ is used to denote the output sample generated by joining the i-th anchor sample with the j-th experiment sample:

- The number of regions in output is estimated as:

$$N_{i \bowtie j} = N_i \cdot s \cdot \delta_j$$

where s is the length of the *search space*, while δ_j is an estimation for the region density of the j-th experiment sample:

$$\delta_j = \frac{N_j}{u_j}$$

- Other features are estimated depending on the `coord-gen` option, as follows:
  - If `coord-gen = LEFT`, then:

$$(w, u)_{i \bowtie j} = (w, u)_i$$

  - If `coord-gen = RIGHT`, then:

$$(w, u)_{i \bowtie j} = (w, u)_j$$

  - If `coord-gen = INT`, then we use the following upper bound:

$$(w, u)_{i \bowtie j} = (min(w_i, w_j), min(u_i, u_j))$$

  - The `CAT` option is much less used in the queries; it requires reprofiling[7].

6. JOIN and MAP can be preceded by a meta-join condition which imposes an equi-join on categorical metadata attributes (e.g. `CancerType, CellLine, Antibody, PatientId`). When the meta-join condition is present, it selects pairs of samples with identical values of the categorical attributes. In this paper, we disregard meta-joins; in GMQL they are computed before JOIN and MAP, and profiles are suitably adapted.

7. Reprofiling is not too expensive as it is performed when data are already loaded in memory.

2.2.3 Map

`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, new values produced by aggregation functions 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 regular structure, called **genomic space**, where each experiment sample is associated with a row, each reference region with a column, and each matrix entry is a single value. 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, `MAP` helps in extracting the most interesting regions out of many candidates.

Example 2.2. Consider the following query:

```
A = SELECT() D1;
E = SELECT() D2;
R = MAP() A E;
```

computed on the same datasets used in Example 2.1. The result, shown in the fourth track of Fig. 1, is made of the reference regions with the count of overlapping experiment regions:

```
R : chromosome, start, stop, strand, count
chr1 100 130 + 1
chr1 260 290 + 2
```

2.2.4 Profiles Estimation for Map Results

The profiles of result samples of a Map operation are simply inherited from the first operand:

$$(N, w, u)_{ij} = (N, w, u)_i$$

3 Optimal Binning

Genome binning refers to the subdivision of the genome into small, identical partitions or **bins**. A similar partitioning method is used by the UCSC Genome Browser [24] in order to speed up the loading of samples within a genome browser window. Binning algorithms are reviewed in the related work section.

The process of binning splits every chromosome of the genome into several bins of equal size b. For each chromosome, bins are progressively numbered starting from 0; the *i-th* bin spans from $b \times i$ to $b \times (i+1) - 1$; a genome position placed at i bases from the chromosome start is assigned to the bin $\beta(i) = \lfloor i/b \rfloor$.

In this section, we consider each map and join operation separately. At execution time, the optimal bin size is computed for each operation, and then the regions of each operand's sample are assigned to the bins. In the next section, we will describe an optimization that applies to chains of join and map operations.

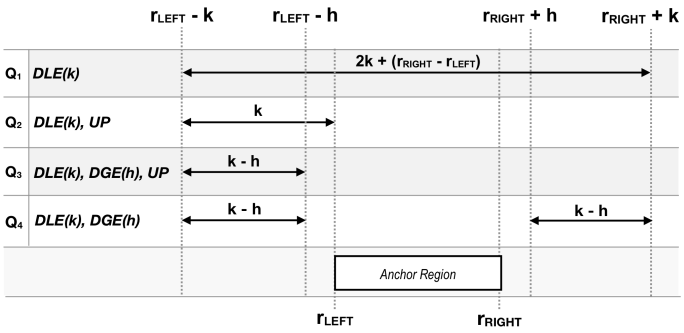

Fig. 2: Four search space topologies

3.1 Optimal Binning for Join

The join implementation strategy, discussed in [23], occurs by first considering the distal conditions of the *first-step clauses* (as defined in Section 2.2.1); this part of the computation is the most expensive as it requires computing all the matching pairs, while the subsequent computation, when present, selects some of them into the result. Therefore, binning optimization is concerned with the search space as defined by the first-step clauses.

The computation of the optimal bin size occurs as follows:

1) First, we estimate how many regions of an experiment sample are assigned to bins - see formula (1).
2) Next, we estimate how many regions of an anchor sample are assigned to bins. Anchor regions are replicated to several bins in order to properly capture the distal conditions by means of region intersection. Depending on the structure of genometric predicates, we obtain four cases. In the first three cases, the estimate is summarized by using formulas which just differ for a term - see formula (2).
3) Next, we discuss the fourth case in which the search space has a gap, and we introduce a correction to the estimate, which has two distinct cases depending on a comparison between the bin size and the length of the gap - see formula (3).
4) Next, we compute the total number of regions that are assigned to bins - see formula (4) - and the total number of pairs of regions produced at all the bins - see formula (5). The join cost is a linear combination of these totals - see formula (7). The rationale of the cost function is that each choice of binning generates a need of replicating regions to bins, evaluated by the first counter, and of joining regions within bins, evaluated by the second counter.
5) Finally, we obtain the optimal bin size by minimizing the cost function. This yields to a single formulation in the first three cases and to four different formulations in the fourth case.

We start with the estimate of data replication depending on the bin size. We denote as $\rho(b)$ the number of regions contained in a binned sample with a bin of size b.

Binning of the experiment dataset produces a copy of the experiment region for each bin that the region intersects. The number of regions in a binned experiment sample can be estimated as:

$$\rho_E(b) = N_E + N_E \cdot \frac{w-1}{b} \quad (1)$$

Intuitively, the binned sample will contain at least the number of regions in the original sample (N_E), plus a number of replicates that depends on the ratio between the region length (w) and the bin size (b).

Anchor regions are replicated to every bin intersecting with its search space. Depending on the genometric predicate provided by the user, the search space assumes 4 different topologies, represented in Fig. 2 and here explained:

- Q_1 : when only `DLE(k)` is present, the search space is the contiguous interval $[r_{left} - k, r_{right} + k]$, where r_{left} and r_{right} denote the coordinates of the left and right ends of the region. With a query of this type, the length of the search space associated to a generic anchor region of length w can be computed as:
$$l_1 = 2 \cdot k + w_A$$
- Q_2 : when only `DLE(k)` and the stream clause `UP` (`DOWN`) are present, the search space is the contiguous interval $[r_{left} - k, r_{left}]$ $([r_{right}, r_{right} + k])$. With a query of this type, the length of the search space associated to a generic anchor region is simply:
$$l_2 = k$$
- Q_3 : when `DLE(k)`, `DGE(h)` and the stream clause `UP` (`DOWN`) are present, the search space is the contiguous interval $[r_{left} - k, r_{left} - h]$ $([r_{right} + h, r_{right} + k])$. With a query of this type, the length of the search space associated to a generic anchor region of length w can be computed as:
$$l_3 = k - h$$
- Q_4 : when `DLE(k)`, `DGE(h)` but no stream clause is present, the search space is no more a single contiguous interval. Indeed, in this case the search space is made of two disjoint symmetric intervals of length l_3, i.e. same length of the previous case, separated by a gap of length $\gamma = 2 \cdot h + w_A$.

If the query is of type $Q_{1|2|3}$, i.e. the search space associated to each anchor region is a single contiguous interval, the number of regions in a binned reference sample can be computed as:

$$\rho_{A_{1|2|3}}(b) = N_A + N_A \cdot \frac{l_{1|2|3} - 1}{b} \quad (2)$$

this equation is similar to equation (1), where the average region length is replaced with the average search space length.

When the query is of type Q_4 the search space is made of two distinct intervals each having length l_3. In general, we could always estimate the number of regions in the binned sample as double the amount of regions estimated when the query is of type Q_3:

$$\rho_{A_4}(b) = 2 \cdot \rho_{A_3}(b)$$

However, when the bin size is greater than the gap length, i.e. when $b > \gamma$, the previous expression counts twice the replicates generated by the search space at the

bin associated to the gap, therefore above that threshold we need a different estimation. However, for $b > \gamma$ the length of the gap can be ignored, i.e. we can assume that the search space is the single contiguous interval $[r_{left} - k, r_{right} + k]$; hence we can estimate the number of regions in a binned anchor sample as we did in case Q_1, i.e.:

$$\rho_{A_4}(b) = \rho_{A_1}(b)$$

Therefore, the amount of regions in a binned anchor sample with a query of type Q_4 is estimated with the following piecewise-defined continuos function:

$$\rho_{A_4}(b) = \begin{cases} 2 \cdot \rho_{A_3}(b) & \text{if } b \leq \gamma \\ \rho_{A_1}(b) & \text{if } b > \gamma \end{cases} \tag{3}$$

that has a derivative discontinuity in γ.

The total number of regions in the binned datasets (both anchor and experiment) can be computed as:

$$\tau_1(b) = \sum_{i=1}^{S_A} \rho_{A_i}(b) + \sum_{i=1}^{S_E} \rho_{E_i}(b) \tag{4}$$

where S_A (S_E) denotes the number of samples in the anchor (experiment) dataset and $\rho_{A_i}(b)$ ($\rho_{E_i}(b)$) is the number of regions in the i-th binned anchor (experiment) sample.

After anchor regions and experiment regions are joined in each bin the overall number of joined couples can be estimated as:

$$\tau_2(b) = \frac{X}{b} \cdot \left(\sum_{i=1}^{S_A} \frac{\rho_{A_i}(b)}{u_{A_i}/b} \cdot \sum_{i=1}^{S_E} \frac{\rho_{E_i}(b)}{u_{E_i}/b} \right) \tag{5}$$

where $\frac{X}{b}$ is the estimation of the total number of bins, $\frac{\rho_{A_i}(b)}{u_{A_i}}$ ($\frac{\rho_{E_i}(b)}{u_{E_i}}$) is an estimation of the amount of regions from the i-th binned anchor (experiment) sample contained in a single bin and S_A (S_E) is the number of anchor (experiment) samples. X estimates the length of the space occupied by both anchor and experiment regions to be joined together and is computed as:

$$X = min\left(max_{i=1..S_A}(u_{A_i}), max_{i=1..S_E}(u_{E_i}) \right) \tag{6}$$

We express the cost function as a linear combination of $\tau_1(b)$ and $\tau_2(b)$:

$$c(b) = \alpha_1 \cdot \tau_1(b) + \alpha_2 \cdot \tau_2(b) \tag{7}$$

where α_1 and α_2 are tuning parameters whose estimation is discussed in Section 6.

The optimal bin size b^* for cases $Q_{1|2|3}$ can be simply found minimizing the cost function, i.e

$$b^*_{1|2|3} = \arg\min_b \quad c(b)$$

and the expression of the optimal bin size is:

$$b^*_{1|2|3} = \left(\frac{\frac{\alpha_1}{\alpha_2} \cdot \sum_{i=1}^{S_A} N_{A_i} \cdot (l_{1|2|3} - 1) + \sum_{i=1}^{S_E} N_{E_i} \cdot (w_{E_i} - 1)}{X \cdot \sum_{i=1}^{S_A} \frac{N_{A_i}}{u_{A_i}} \cdot \sum_{i=1}^{S_E} \frac{N_{E_i}}{u_{E_i}}} + \right.$$
$$\left. + \frac{\sum_{i=1}^{S_A} \frac{N_{A_i} \cdot (l_{1|2|3} - 1)}{u_{A_i}} \cdot \sum_{i=1}^{S_E} \frac{N_{E_i} \cdot (w_{E_i} - 1)}{u_{E_i}}}{\sum_{i=1}^{S_A} \frac{N_{A_i}}{u_{A_i}} \cdot \sum_{i=1}^{S_E} \frac{N_{E_i}}{u_{E_i}}} \right)^{1/2} \tag{8}$$

For case Q_4 , the optimal bin size is computed in the following way:

- if $b^*_3 \leq \gamma$ and $b^*_1 \geq \gamma$: the minimum is:

$$b^*_4 = \begin{cases} b^*_3 & if \quad c_3(b^*_3) < c_1(b^*_1) \\ b^*_1 & if \quad c_3(b^*_3) \geq c_1(b^*_1) \end{cases}$$

- if $b^*_3 > \gamma$ and $b^*_1 \geq \gamma$: the minimum is:

$$b^*_4 = b^*_1$$

- if $b^*_3 \leq \gamma$ and $b^*_1 < \gamma$: the minimum is:

$$b^*_4 = b^*_3$$

- if $b^*_3 > \gamma$ and $b^*_1 < \gamma$: this is a limit case since the minimum equals the critical point:

$$b^*_4 = \gamma$$

3.2 Optimal Binning for Map

The computation of the optimal bin size for a map operation occurs as follows:

1) First, we estimate how many regions of reference and experiment datasets are produced by binning. Such computation is simple - same as formula (1) for join.
2) Next, we compute the total number of regions that are assigned to bins - same as formula (4) for join - and the complexity of sorting and comparing regions - see formula (10). The map cost is obtained as a linear combination of these totals - see formula (12). Similarly to the Join, the rationale of the cost function is that each choice of binning generates a need of replicating regions to bins, evaluated by the first counter, and of comparing regions within bins, evaluated by the second counter.
3) Finally, we obtain the optimal bin size by minimizing the cost function.

Evaluating data replication for a Map is simple: both reference and experiment regions are replicated in the bins they intersect. Therefore, the number of regions in binned reference and experiment samples is estimated in both cases as in (1 for join, i.e.:

$$\rho_{R_i}(b) = N_{R_i} + N_{R_i} \cdot \frac{w_{R_i} - 1}{b}$$

$$\rho_{E_i}(b) = N_{E_i} + N_{E_i} \cdot \frac{w_{E_i} - 1}{b}$$

and the total number of regions in the binned datasets (both reference and experiment) can still be estimated as in (4) for join, i.e.:

$$\tau_1(b) = \sum_{i=1}^{S_R} \rho_{R_i}(b) + \sum_{i=1}^{S_E} \rho_{E_i}(b) \tag{9}$$

Compared with join, the map operation is simpler; the algorithm used to compare reference and experiment regions in each bin, described in [23], uses merge-sort, whose complexity, as a function of the bin size, can be approximated by:

$$\tau_2(b) = \frac{X}{b} \cdot \left[\left(\sum_{i=1}^{S_R} \frac{\rho_{R_i}(b)}{u_{R_i}/b} \right) \cdot \log_2 \sum_{j=1}^{S_R} \frac{\rho_{R_i}(b)}{u_{R_i}/b} + \sum_{i=1}^{S_E} \frac{\rho_{E_i}(b)}{u_{E_i}/b} \cdot \log_2 \frac{\rho_{E_i}(b)}{u_{E_i}/b} \right] \quad (10)$$

where $\frac{X}{b}$ is an estimation of the total number of bins, $\left(\sum_{i=1}^{S_R} \frac{\rho_{R_i}(b)}{u_{R_i}/b} \right) \cdot \log_2 \sum_{j=1}^{S_R} \frac{\rho_{R_i}(b)}{u_{R_i}/b}$ is the complexity of sorting reference regions, that are sorted within each bin independently on their sample, and $\sum_{i=1}^{S_E} \frac{\rho_{E_i}(b)}{u_{E_i}/b} \cdot \log_2 \frac{\rho_{E_i}(b)}{u_{E_i}/b}$ is the complexity of sorting experiment regions sample by sample within each bin. X is again computed as:

$$X = min\left(max_{i=1..S_R}(u_{R_i}), max_{i=1..S_E}(u_{E_i}) \right) \quad (11)$$

The cost function is again a linear combination of $\tau_1(b)$ and $\tau_2(b)$:

$$c(b) = \alpha_1 \cdot \tau_1(b) + \alpha_2 \cdot \tau_2(b) \quad (12)$$

where α_1 and α_2 are tuning parameters.

Again, finding the optimal bin size b^* means looking for the b that solves the minimization equation:

$$\frac{dc(b)}{db} = 0$$

.

The development of the last equation, properly rearranged and approximated, is an equation of type:

$$\alpha_2 h_1 \cdot \log b + \alpha_2 h_2 \cdot b + (\alpha_2 h_3 - \alpha_1 h_4) = 0 \quad (13)$$

where $h_{1|2|3|4}$ are constant terms derived from the dataset profiles. As argued in the Appendix, there is no elementary solution for (13), but we can resort to an expression of the optimal bin size that uses the Lambert-W function, an ubiquitous mathematical tool [53]. Denoting with W the Lambert W function, the solution of (13) can be expressed as:

$$b^* = \frac{h_1}{h_2} \cdot W\left(\frac{h_2}{h_1} \cdot e^{\frac{\alpha_1}{\alpha_2} \frac{h_4}{h_1} - \frac{h_3}{h_1}} \right)$$

4 Experiments

4.1 Join

We validated the model by means both of synthetic and of real datasets; synthetic datasets were generated by starting from profiles, while real datasets were extracted from genomic repositories. In general, all predictions are very accurate, and show that the cost model is extremely precise in predicting the execution time of computations, both with synthetic and with real datasets. As the cost model does not include the prediction of constant terms, we visualize the predicted cost curve so that costs and times are referred to the same scale, by setting the predicted cost at the optimal predicted bin size to be identical to the execution time for that bin size; we then repeat the testing for variable bin sizes and plot the corresponding execution times against the estimated cost; continuous lines represent the estimated costs and dashed lines represent the measured execution times, obtained by connecting the discrete execution times measured for given bin sizes. A red square marker denotes the position of the predicted optimal bin size for a query.. Tests were executed on an Amazon EMR Cluster made of one master node and five instances of type m3.2xlarge; execution times are expressed in seconds.

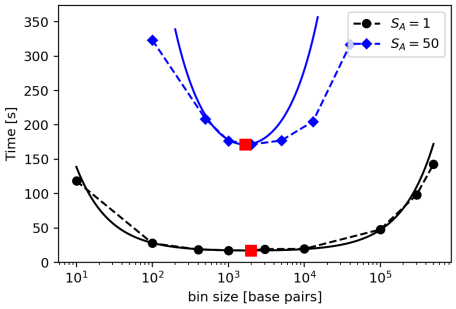

Fig. 3: Comparison of real and predicted execution time and optimal bin size for the JOIN operation on synthetic datasets that differ in the number of anchor samples S_A.

TABLE 2: Profile features used as baseline

Feature	Value
S_A	1
S_E	5
N	$25 \cdot 10^4$
w	100
u	$5 \cdot 10^7$

Synthetic data were generated by considering a single interval of regions and a profile baseline shown in Table 2; region lengths were generated by using a Gaussian distribution centered on the mean value w. We then changed the baseline by one dimension at a time, diagrams 3 to 5 illustrate interesting cases associated to three dimensions.

Figures 3 and 4 show that by rising the number of anchor and of experiment samples the optimal bin size does not change, as expected. However, the execution time is significantly different. They also show that a correct estimate of the optimal bin size is most relevant with larger datasets, as the curves (both in the estimated and computed cases) show a significant slope. In Fig. 3, where the optimal bin size corresponds to about 3000 bases, a choice of bin size of 50 vs 5000 bases yields to doubling execution; largest bin sizes correspond to extremely inefficient or possibly non-terminating executions.

Figure 5 shows that the optimal bin size changes as a function of the change in distal conditions. It also shows the most dramatic change in execution time, as the execution time associated with the optimal binning and a distance of 500 is about 10 seconds whereas the execution time with a distance of $50,000$ rises up to about 10 minutes. Instead, execution time is less influenced by the average region length when it is below the bin size, and similarly it is less influenced by the number of regions (the corresponding diagrams are omitted).

Figure 6 shows two different join executions, obtained by varying three parameters: the number of anchor and experiment samples and the search space. The figure shows

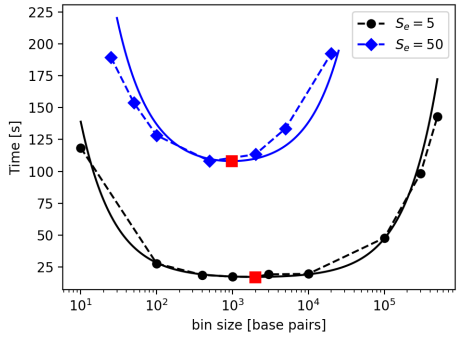

Fig. 4: Comparison of real and predicted execution time and optimal bin size for the JOIN operation on synthetic datasets that differ in the number of experiment samples S_E.

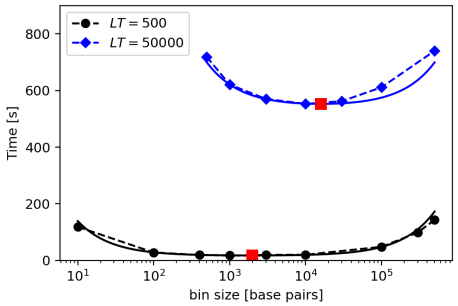

Fig. 5: Comparison of real and predicted execution time and optimal bin size for the JOIN operation on synthetic datasets that differ in the size of the search space.

that execution time does not differ too much in the interval between the two optimal bin sizes (700 and 8000 bases respectively), however a wrong choice of bin size easily brings to suboptimal executions. In particular, by setting the bin size to 200, execution time for the $(1, 10, 50000)$ case triples w.r.t the optimum; by setting the bin size to 30000, execution time for the $(10, 10, 500)$ case doubles.

In Section 3.1 we discussed the cost function of case Q_4 , which has a critical point in correspondence with the derivative discontinuity of the estimate reported in formula 3. Figure 7 experimentally confirms the model, showing that the discontinuity occurs at bin size 1200 - which separates the experimental curve into two parts. In the experiment, other parameters are: $h = 500$ and $w = 100$.

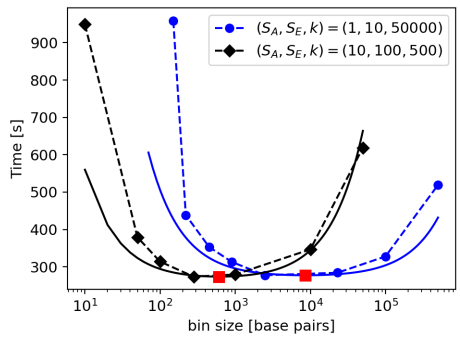

Fig. 6: Two different JOIN executions show different optimality ranges for the bin size.

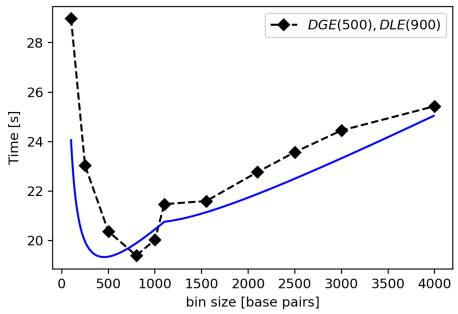

Fig. 7: Comparison of real and predicted execution times for a query of topology Q_4; note that the cost function has a discontinuity at $\gamma = 2 \cdot h + w$, confirmed by the real case.

TABLE 3: Datasets' features . S, N and w are averaged over the number of samples.

	Anchor	Experiment
S	15	30
N	$6.7 \cdot 10^4$	$1.8 \cdot 10^5$
w	$6.1 \cdot 10^4$	$2.3 \cdot 10^3$
u	$2.49 \cdot 10^8$	$2.49 \cdot 10^8$

The real-life experiment was performed on large datasets (450 sample pairs, each with 120 billion region pairs) and by using a very large search space of 10^6 bases, so as to stress-test our system (recall the discussion relative to Fig. 5). We used replicas of the RefSeq genes as anchor and NarrowPeak ChIPseq datasets from Encode as experiments, the corresponding profiles are shown in Table 3. The test in Fig. 8 reports execution times in minutes; also in this case, the estimate is very accurate, and in particular the optimal bin size is almost identical in the two cases.

4.2 Map

Execution times of the map operation exhibit a steep descent towards the minimum, followed by linear growth in logarithmic scale after the minimum, which is due to the presence of a logarithmic complexity in the cost function. In this case, the underestimation of the optimal bin size causes a strong rise of execution times, while its overestimation is less critical.

In particular, Fig. 9 shows a shift in the cost function with the growth of the average length of regions (from $w = 10$

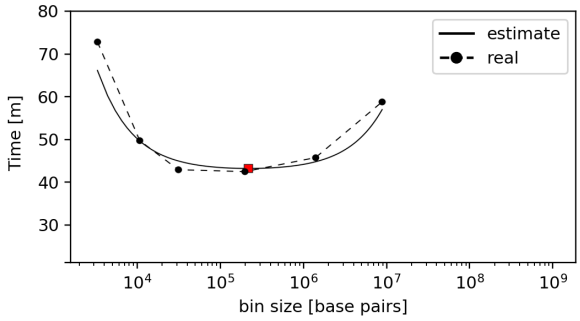

Fig. 8: Comparison of real and predicted execution time and optimal bin size for the JOIN operation on real datasets.

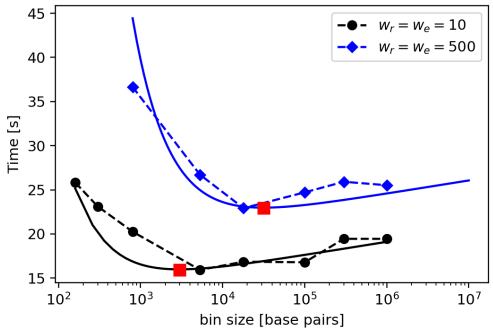

Fig. 9: Comparison of real and predicted execution time and optimal bin size for the MAP operation on synthetic datasets that differ in the average length of regions w.

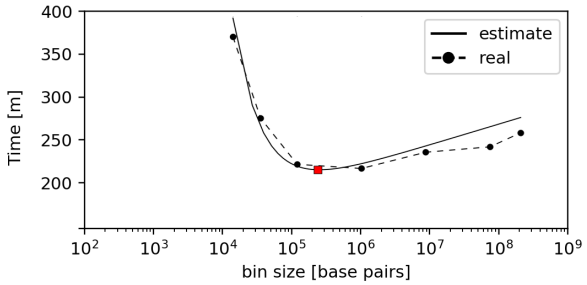

Fig. 10: Comparison of real and predicted execution time and optimal bin size for the MAP operation on real datasets.

to $w = 500$); the shift causes the optimal bin size also to increase (from 500 to 5000 bins), while in a join a difference of average length of regions does not significantly change the optimal bin size. As in join, execution time is influenced by the number of reference and experiment samples, but they do not cause significant changes of the optimal bin size (diagrams omitted).

The real-life experiment was performed on the profiles of Table 3. The test in Fig. 10 reports execution times in seconds, as the corresponding map operation is not as complex as the join (in particular, anchor regions in the map are two orders of magnitude smaller the join search space of 10^6 bases); the estimate is also in this case very accurate, and in particular the optimal bin size is almost identical in the two cases.

5 Optimizing Binning over Sequences of Operations

So far we have discussed the individual optimization of binning algorithms for two different domain specific operations, JOIN and MAP, but most conjunctive queries include several such operations. In such case, binning optimization may take into account several operations at the same time; in particular, the most interesting option consists of keeping the same binning of operands of consecutive operations. In this section, we discuss when such option is possible (given the semantics of operations) and convenient (as it incurs less costs).

A GMQL query can be represented as a Directed Acyclic Graph (DAG) in which vertices represent operations and edges represent dependencies among operations. In our representation, an edge from v to u implies that the output dataset of operation v is the input dataset of operation u. Binary operations, such as JOIN and MAP, have two input edges and one output edge. An example is shown in Figure 11, representing the DAG of the following query:

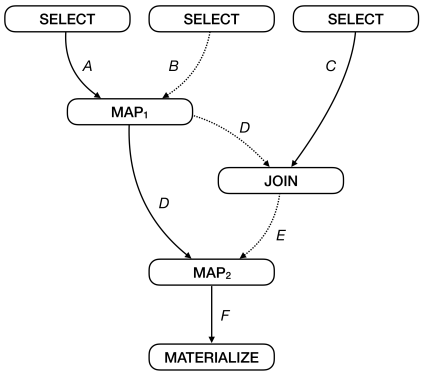

Fig. 11: DAG of a GMQL query; dashed edges denote experiment datasets. Edges are labeled with the variable (dataset) identifier.

```
A = SELECT() FROM DS1;
B = SELECT() FROM DS2;
C = SELECT() FROM DS3;
D = MAP() A B;
E = JOIN( DLE(100); RIGHT) C D;
F = MAP() D E;
MTERIALIZE F INTO RES;
```

We can apply two different kinds of multi-operation optimizations:

- *Binning Reuse*: when the same dataset is in input to several operations, we can use the same binning for both of them.
- *Binning Chaining*: when a dataset, produced as output of a binary operation, is used in a following binary operation, we can use for second operation the binning produced by the first one, without rebinning.

From a semantic point of view, bin chaining opportunities are summarized in Figure 12; it shows that:

- JOIN can get as input binned results for the experiment dataset produced by a MAP or JOIN with RIGHT composition option: the anchor can be chained only in the case when the producer operation is a JOIN with LEFT composition option and the search space is identical.
- MAP can get as input binned results both for anchor and experiments produced by either a MAP and or by a JOIN with RIGHT composition option.

Figure 11 shows examples of both reuse and chaining: (i) the dataset D, resulting after the MAP_1 operation, is in input to the JOIN and next chained to the MAP_2 operation, hence its binning can be reused by the MAP_2 operation; (ii) the datasets D and E, produced in output by MAP_1 and JOIN with RIGHT composition option respectively, are in input to the MAP_2, hence their binning can be chained.

The savings which can be obtained thanks to binning reuse or chaining are essentially related to one of the two

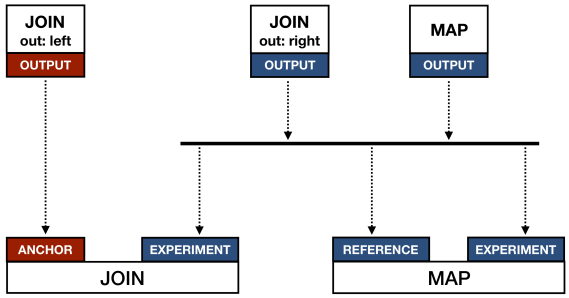

Fig. 12: Bin chaining with sequences of JOIN and MAP operations.

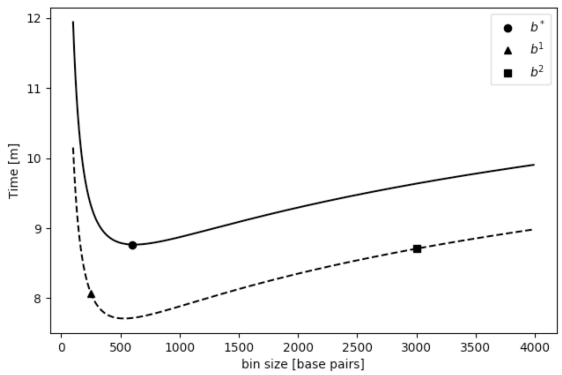

Fig. 13: Bin choices with sequences of JOIN and MAP operations.

terms counting the number of regions over the anchor (first term) or experiments (second term) appearing in formula (4) (for join) or (9) (for map). These are weighted by tuning parameters α_1, where α_1 and α_2 essentially trade data shuffling and computational costs, in the overall cost formulas (7) (for join) and (12) (for map). Thus the reduction of costs incurring by reuse or chaining is properly computed by omitting the appropriate term of formulas (4) or (9) weighted by α_1 - let's denote such term as τ_-. Hence, decision about reusing/chaining a bin size b^+ instead of the optimal bin b^* requires the following term to be positive:

$$c(b^+) - c(b^*) - \tau_-(b^+)$$

Figure 13 shows that the binning for MAP_2 should consider 3 options: (a) the optimal bin b^*; (b) the binning b^1 that is carried by the input D, in its double role of being reused or chained, (c) the binning b^2 that is carried by the input E. The dashed curve shows the saving in the cost function when the binning of the experiment dataset is omitted; the optimal bin is b^1 and the resulting saving over b^*, confirmed by experiments, is about 12% .

6 Cluster Execution

6.1 GDMS Deployment

The Genomic Data Management System (GDMS) manages the execution of GQML queries over a single or several machines providing different interfaces to the user (web interface and web services, command-line interface and several programmng language-specific APIs). The most up-to-date

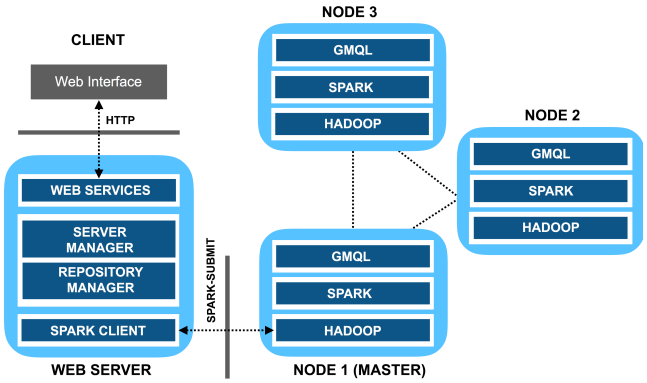

Fig. 14: Deployment of GDMS on a cluster provided by Cineca.

implementation of GMQL is based on Apache Spark and uses the Hadoop Distributed File System (HDFS) for parallel execution over multiple machines. Our publicly accessible version of the GDMS is deployed on a cluster provided by the CINECA consortium according to the schema depicted in Fig. 14, where the main components are:

- Cluster: made of three nodes (each having 40 vCPUs, 125GB RAM and 3TB disk) equipped with Spark and Hadoop running on Ubuntu Xenial. One of the nodes (labeled as master) receives submissions of the GMQL command-line interface JAR with the query as its input and drives the execution until results are stored back to the HDFS.
- Web Server: provides web services and a web interface to the users, transforming their query execution requests into job submissions to the cluster. Moreover, the Web Server, through a component called Repository Manager, manages the public and private datasets stored in the HDFS.

The system is easily portable to other cloud computing infrastructures such as Amazon AWS, using any of the latest emr releases supporting Spark 2.2.x ; in the simplest case all you need to do is to submit the GMQL command-line interface JAR to Spark with the desired data and query as input.

6.2 Model Fitting

The two cost functions defined in the previous sections showed the presence of some tuning parameters, namely α_1 and α_2, that differ depending on the operation. These parameter approximate, respectively, the time employed to transform and shuffle a single binned region (α_1) and the time spent to compare a couple of binned regions (α_2). The processing time, on its turn, depends on several factors, mostly related to the configuration of Spark and to the hardware used to run each application (CPU speed, RAM, number of cluster nodes, network speed and so on). Since the number of factors that should be taken into account is too large to build a separate analytical model, α_1 and α_2 are treated as parameters to be tuned in each experimental setting; we re-tune the parameters only when we significantly change the configuration of Spark and/or the configuration of the cluster. Our method used for parameter tuning is

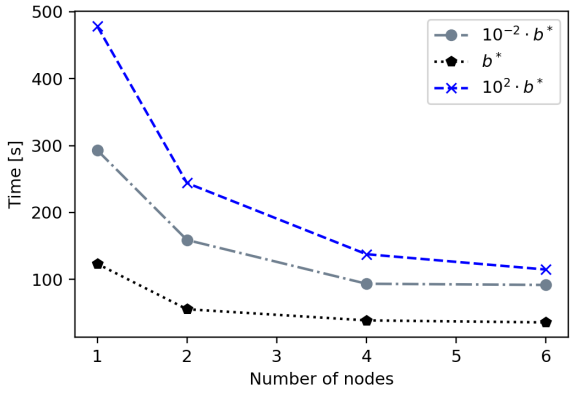

Fig. 15: Scaling of execution time by increasing the number of core nodes, using optimal and suboptimal bin sizes.

based on gradient descent. Since our model is not designed to predict the exact execution times of the cost function, but rather its shape (the correct match would be only on the first derivative), we compute the residuals on a normalized cost function $c_N(b)$ which is next defined.

Given an experimental execution consisting of n points (data pairs) (b_i, e_i), $i = 1...n$, where e_i is the execution time observed at b_i, the normalized cost function $c_N(b)$ is:

$$c_N(b) = c(b) - min\ c(b) + min(e_1, e_2, ..., e_n)$$

The optimal values of α_1 and α_2 are those minimizing the following function:

$$J(\alpha_1, \alpha_2) = \frac{1}{2n} \sum_{i=1}^{n} \left(e_i - c_N(b_i, \alpha_1, \alpha_2)\right)^2$$

For join, the initial values for α_1 and α_2, respectively 10^{-6} and 10^{-8}, were computed greedily. Gradient descent is applied separately to different executions of the same operator, changing data profiles and query options, on the same execution environment, then the obtained optimal values for α_1 and α_2 are averaged. Table 4 shows the values of α_1 and α_2 computed for different environments; for those experiments Spark was configured to maximize the usage of available resources.

TABLE 4: Values of α_1 and α_2 for different environments (join).

Site	(v) Cores	RAM [GB]	Nodes	α_1	α_2
AWS	16 gib	61	1	$1.5 \cdot 10^{-6}$	$7.5 \cdot 10^{-8}$
			2	$1.0 \cdot 10^{-6}$	$4.1 \cdot 10^{-8}$
			4	$0.6 \cdot 10^{-6}$	$2.0 \cdot 10^{-8}$
			6	$0.5 \cdot 10^{-6}$	$1.5 \cdot 10^{-8}$
Cineca	24	80	3	$0.4 \cdot 10^{-6}$	$1.2 \cdot 10^{-8}$
Polimi	25	256	1	$0.6 \cdot 10^{-6}$	$2.3 \cdot 10^{-8}$

6.3 Scaling

In Fig. 15 we show how the execution time of a Join operation scales increasing the number of nodes of a cluster for optimal and suboptimal bin sizes. The experiments were performed on a AWS cluster made of core instances of type *r3.2xlarge* (16 vCores, 61 GiB memory) with the *maximizeResourceAllocation* option set to *true*, so to maximize the resources allocated for the job. In the experiment, we join one reference sample with forty experiment samples, setting `DLE(1000)` as genometric predicate; each sample contains $200 \cdot 10^3$ regions (average region length 300 bases) distributed on a useful space of 10^8 bases (interpreted as a single chromosome); hence, potentially matching pairs are $1,600,000 \cdot 10^6$. The shorter execution time is about 40s, obtained with 6 nodes and an optimal bin size around $3 \cdot 10^3$ bases. While adding nodes reduces execution times, it also brings higher costs; in in our example, execution time with 2 nodes and optimal bin size is 55s, which can be considered as acceptable for this query.

Suboptimal solutions in the bin size increase the amount of processing, and more nodes are required to make the execution time stable. As expected, the scaling of suboptimal solutions "follows" the scaling of the optimal solution, meaning that the ratio between suboptimal and optimal execution times remains constant independently on the number of nodes. Therefore, the difference in time with respect to an optimal solution becomes more remarkable when less resources are available; optimal binning is then more effective when GMQL runs locally or on a small cluster. Moreover, here we can see that, with the same scaling factor (10^2), a smaller bin size takes more time than a bigger bin size. In general, smaller bin sizes increase replication so that more data are present in memory and must be processed by the operations preceding the final comparisons. Bigger bin sizes, instead, minimize the replication, increasing the number of final comparisons to perform; therefore they take less memory but increase the CPU workload; depending on the availability of cluster resources, bigger bin size are preferable (like in this case).

7 Related Work

Several cloud-based systems are concerned with the management of secondary analysis pipelines [2], [21] and several cloud-based libraries provide the methods for those pipelines [35] [31] [46]. Effective metadata management for selecting samples using key-based NoSQL storage for referring to genomic datasets is described in [50]. We next concentrate on region processing, the most critical aspect of ternary data management, and specifically with the technology used for massive region-based processing, with the libraries supporting low-level region operations and with high-level query languages.

7.1 Technologies for region processing

Genomic join and map operations require handling thousands of reference and experiment samples, in turn consisting of thousands of regions spanning over the whole genome. Genomic region management could be considered as a special case of spatial or temporal data management, for which several scalable solutions exist. Among them, several architectures show extremely good scalability, including Simba [47], GeoSpark [49], Hadoop-GIS [3], Spatial Hadoop [19], and Spatial Spark [48]. All these tools support range query, one point KNN, and spatial join of two datasets, while only Simba supports general KNN join. Internally, they use a mix between R-Trees and map reduce algorithm,

where data are partitioned based on initial scan or sampling, then local index and global indexes are built [19], [27].

In comparison to spatial data, genomic data show three main differences: (1) while in spatial data most queries are geo-referenced and look for proximal data to a given point, in genomics queries typically consider the whole genome, with no notion of locality; (2) the arbitrary composition of distal conditions in join clauses makes a direct use of indexing schemes not applicable in finding all cases of matching intervals (3) the same query can include several cascaded join and map operations. Therefore, methods of spatial data management contribute relevant background but they cannot be directly applied, and binning is instead commended.

Binning algorithms partition the join operands in order to speed up the evaluation; the concept of binning was introduced in [33], and genome binning are used in the UCSD genome browser to speed up the loading of genomic regions to the browser [24]; Afrati et al. [4] analyzed binning-based algorithm in order to assess computation bounds. In [18] the authors propose an algorithm based on data binning; our algorithms differ from their proposal in the way we check the intersection and avoid output duplicates. In our earlier work [23] we explained the binning methods but we did not discuss the optimal binning; therefore, [23] is an important background of this paper, but the results reported in this paper are new.

A line sweep algorithm is implemented in BEDOPs and BedTOOLS [29], [34] to compare two files (corresponding to our reference and experiment files), by sorting them on the start of the interval and then sweeping the two files sequentially, comparing the intervals and finding the intersections. BEDTools incorporates the genome-binning algorithm used by the UCSC Genome Browser in the search for overlapping regions; in [15] we show that region intersection between two samples in GMQL has slightly better performance than BEDTools. GMQLs use of binning is much more complex from a system's perspective, as it is implemented in the cloud environment to support implicit iteration over thousands of sample pairs.

We recently proposed bi-dimensional binning as an alternative to mono-dimensional binning; with such approach, every region is modeled as a point in a bidimensional space, and join or map condition can be expressed as intersection of points with suitable arrays. The method has been applied to a SciDB implementation of GMQL, as discussed in [14], showing some improvement of performance; in our future work, we plan to test the feasibility of the bi-dimensional approach with Spark.

7.2 Libraries for region management

BEDTools and BEDOPS apply to the BED format, i.e. to a format based on regions similar to GDM; a functional comparison of these tools with GMQL is published as supplemental material to [28]. They can be used from within software environments for bioinformatics (e.g., *BioPerl*, *BioPython*, *R* and *Bioconductor*), but are not designed for cloud computing.

Other works have proposed the embedding of query processing functions within libraries that can be integrated within programs. In particular, [32] presents a rather elegant mathematical formalism, based on set algebra, delivered as the *Genomic Region Operation Kit* (GROK) library. In comparison, GROK supports lower-level abstractions than GMQL and some low-level operations (e.g., flipping regions) that are not directly supported by GMQL, but they must be embedded into C++ programming language code. GROK is unsuitable for parallelization and does not deal with metadata. Recently, BigDataScript [17] was developed; it embeds some high-level relational concepts.

7.3 High-level Query Languages

SciDB is an open-source system for scientific data management supporting an add-on specifically dedicated to tertiary data analysis [7], [12]. SciDB queries are programmed using the Array Functional Language (AFL), a query language where each operation is defined as a function that receives as input either one or two arrays and produces as output one array; operations can be nested. The database engine of SciDB is based on the array data model; it is designed to work on a cluster of nodes, exploiting data distribution and parallelism. Arrays are divided into chunks; an hash function uses the dimension values associated to each chunk in order to assign it to a specific node of the cluster; by using this method, called *Multidimensional Array Clustering*, every query processing operation is mapped to specific chunks and executed in parallel at the nodes where such chunks are allocated. A comparison of SciDB with our Spark-based implementation is provided in [13]; our binning-based join implementation has better performances, whereas SciDB appears superior for selection and aggregation operations which take advantage of the array-based architecture.

Deepblue [5] is the query language used for accessing the BLUEPRINT data repository; the language expressions combine metadata and region-based predicates and constructors, although with a limited expressive power when compared to GMQL.

Several query languages compare more closely to GMQL as they introduce a relational paradigm into genomic computing; among them, [36] [10] [40] [52]; none of them deals with metadata thereby lacking the ability to constructively assign metadata to query results, a distinguishing feature of GMQL. Genomic Query Language (GQL), presented in [10], [25], applies to raw data and includes genomic feature calling this approach creates reproducibility issues when compared to more conventional pipelines. Signal Track Query Language (STQL), presented in [52], is instead focused on ternary analysis, hence is closer in design to GMQL. STQL has a rich set of relational operators and highly expressive predicate calculus on distal conditions; it is directly implemented using map-reduce and Hive over Hadoop1, whereas GMQL is implemented on Spark over Hadoop2. GMQL adds to the query language several APIs, repositories and language interfaces, whereas STQL execution in [52] is described from within a Web-based interface.

8 Conclusion

In this paper, we presented the mathematical formulation of the optimal binning problem for join and map operations on genomic regions; we validated the model through a rich set of experiments that show the model's accuracy and sensitivity to the variation of query parameters, using both synthetic and real datasets; we stressed the join operation at work with very demanding parameters (450 sample pairs, each with 120 billion region pairs tested for minimal distance of 10 million bases).

We demonstrated with several examples that execution time doubles with a relatively small error in the choice of bin size, and a big error may even lead to cases where execution cannot be completed, as resources are exhausted. We also discussed the optimization of sequences of operations, showing that binning chaining or reuse can introduce significant savings (in the example we discussed, order of 12%). Finally, we discussed the deployment of GMQL to two cloud environments and discussed the estimation of the parameters α_1 and α_2, whose ratio influences the determination of the optimal bin size. Full GMQL (with metadata and region support) is deployed in a public network at Cineca[8]. We are planning an incubation of the GMQL project within Apache, so as to provide a strong community of users and developers.

We are using GMQL in advanced biological research and for deploying services to be used by the community of biologists and clinicians; discussions are ongoing (as of January 2018) for the inclusion of GMQL within the tools supported by FireCloud [21], a genomic data processing system integrated with GoogleStore, hosted by the Broad Institute. We are also developing a large repository of public data, constructed by integrating and curating data from ENCODE [20], TCGA [45] and other sources; our first step in this direction is the consolidation of an integrative conceptual model for the relevant metadata attributes of several public sources, presented in [11].

Acknowledgments

This research is supported by the ERC Advanced Grant *GeCo* (Data-Driven Genomic Computing), 2016-2021.

References

[1] 1000 Genomes Consortium. An integrated map of genetic variation from 1,092 human genomes. *Nature*, 491, 56-65, November 2012.
[2] Adam. http://www.bdgenomics.org/
[3] A. Ablimit et al. Hadoop gis: a high performance spatial data warehousing system over mapreduce. *Proceedings VLDB*, 6.11 (2013): 1009-1020.
[4] F. Afrati and J. Ullman. Bounds for Overlapping Interval Join of Map Reduce. *Workshop Proceedings, EDBT/ICDT*, 2015.
[5] F. Albrecht et al. DeepBlue Epigenomic Data Server: Programmatic Data Retrieval and Analysis of the Epigenome. *Nucleid Acids Research*, 44/W1, 2016.
[6] A. Alekseyenko et al. Nested Containment List (NCList): a new algorithm for accelerating interval query of genome alignment and interval databases. *Bioinformatics* 23.11, 1386-1393, 2007.
[7] Anonymous paper, Accelerating Bioinformatics Research with New Software gor Big Data to Knowledge (BD2K), Paradigm4 Inc., 2015 (downloaded from: www.paradigm4.com).

8. http://www.bioinformatics.deib.polimi.it/GMQL/interfaces/

[8] Apache Flink. http://flink.apache.org/
[9] Apache Spark. http://spark.apache.org/
[10] V. Bafna et al. Abstractions for genomics.*Commun. ACM*, 56(1):83-93, 2013.
[11] A. Bernasconi et al., Conceptual Modeling for Genomics: Building an Integrated Repository of Open Data. in *Proc. Conceptual Modeling (ER) Conference*, Valencia, Nov. 2017.
[12] P. G. Brown. Overview of SciDB: large scale array storage, processing and analysis. *Proc. ACM-SIGMOD*, 963-968, 2010.
[13] S. Cattani et al. Evaluating big data genomic applications on SciDB and Spark. *Proc. Web Engineering Conference*, Rome, IT, 2017.
[14] S. Cattani et al. P. Bi-Dimensional Binning for Big Genomic Datasets. *Proc. Beyond Map Reduce Workshop*, co-located with ACM-Sigmod, Boston, May 2017.
[15] S. Ceri at al. Data management for heterogeneous genomic datasets. *IEEE/ACM Transactions on Computational Biology and Bioinformatics* DOI: 10.1109/TCBB.2016.2576447, 2016.
[16] 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.
[17] P. Cingolani et al. BigDataScript: a scripting language for data pipelines. *Bio-informatics*, 31(1), 10-16, 2015.
[18] B. Chawda. Processing Interval Joins On Map-Reduce. *Proc. EDBT*, 463-474, 2014.
[19] A. Eldawy SpatialHadoop: A MapReduce framework for spatial data. *Proc. 31st Conference on Data Engineering (ICDE)*, 2015.
[20] ENCODE Project Consortium. An integrated encyclopedia of DNA elements in the human genome. *Nature*, 489(7414):57-74, 2012.
[21] FireCloud. https://software.broadinstitute.org/firecloud.
[22] Google Genomics Could Platform. https://cloud.google.com/genomics/
[23] A. Kaitoua et al. Framework for Supporting Genomic Operations, *IEEE-TC*, 10.1109/TC.2016.2603980, 2016.
[24] W.J. Kent, The human genome browser at UCSC. *Genome Res.*, 2002 Jun;12(6):996-1006.
[25] C. Kozanitis et al. Using Genome Query Language to uncover genetic variation. *Bioinformatics* 30(1):1-8, 2014.
[26] R. M. Layer et al. Binary Interval Search: a scalable algorithm for counting interval intersections, *Bioinformatics* ,29 (1), p1-7, 2013.
[27] M.-L. Lo. Spatial hash-joins. *ACM SIGMOD Record* Vol. 25 No. 2, 1996.
[28] M. Masseroli et al. GenoMetric Query Language: A novel approach to large-scale genomic data management. *Bioinformatics*, doi: 10.1093/bioinformatics/btv048, 2015.
[29] S. Neph, et al. BEDOPS: high-performance genomic feature operations. *Bioinformatics*, 28(14):1919-1920, 2012.
[30] NIH National Human Genome Research Institute, DNA Sequencing Costs. http://www.genome.gov/sequencingcosts/
[31] H. Nordberg et al. BioPig: a Hadoop-based analytic toolkit for large-scale sequence data. *Bioinformatics*, 29(23):3014-3019, 2013.
[32] 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.
[33] J. M. Patel et al. Partition based spatial-merge join. *ACM SIGMOD Record*, 25:2, 1996.
[34] A. R. Quinlan and I. M. Hall. BEDTools: a flexible suite of utilities for comparing genomic features. *Bioinformatics*, 26(6):841-842, 2010.
[35] A. Roy et al. Massively Parallel Processing of Whole Genome Sequence Data: An In-Depth Performance Study, *Proc. ACM-Sigmod*, May 2017, Chicago.
[36] U. Rohm and J. Blakeley. Data management for high-throughput genomics. In *Proc. CDIR*, 1-10, 2009.
[37] A. Schumacher et al. SeqPig: simple and scalable scripting for large sequencing data sets in Hadoop. *Bioinformatics*, 30(1):119-120, 2014.
[38] K. Shvachko et al. The Hadoop distributed file system. *Proc. MSST*, 1-10, 2010.
[39] Z. D. Stephens et Al.: Big Data: Astronomical or Genomical? *PLoS Biol* 13(7), 2015.
[40] S. Tata et al. Periscope/SQL: Interactive exploration of biological sequence databases. *Proc. VLDB*, 1406-1409, 2007.
[41] R. C. Taylor. An overview of the Hadoop MapReduce HBase framework and its current applications in bioinformatics.*BMC Bioinformatics*, 11(12):S1, 2010.

[42] J. Yu, GeoSpark: "A Cluster Computing Framework for Processing Large-Scale Spatial Data". *Proc. ACM SIGSPATIAL GIS 2015*, Seattle, 2015.
[43] R. Xin et al. Shark: SQL and Rich Analytics at Scale. *Proc. ACM-SIGMOD*, 2013.
[44] W. Yu et al. Virtual Shuffling for Efficient Data Movement in MapReduce. *IEEE Transactions on Computers*, 64(2), 2015, doi:10.1109/TC.2013.216.
[45] J. N. Weinstein et al. The Cancer Genome Atlas Pan-Cancer analysis project. *Nat Genet*., 45(10):1113-1120, 2013.
[46] 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.
[47] D. Xie et al. Simba: Efficient in-memory spatial analytics. *Proc. ACM-Sigmod*, 2016.
[48] S. You et al. Large-scale spatial join query processing in cloud. *Data Engineering Workshops (ICDEW)*, 2015.
[49] J. Yu et al.Geospark: A cluster computing framework for processing large-scale spatial data. *Proc. 23rd SIGSPATIAL Conf. on Advances in Geographic Information Systems.* ACM, 2015.
[50] S. Wang et al. High dimensional biological data retrieval optimization with NoSQL technology. *BMC Genomics*, 15(Suppl 8), S3.
[51] M. Zaharia et al. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. *Proc. USENIX*, 15-28, 2012.
[52] Zhu, X. et al. (2017) START: a system for flexible analysis of hundreds of genomic signal tracks in few lines of SQL-like queries. *BMC Genomics*, 18(1), 749.
[53] Katsimpiri C. et al. (2016) The Ubiquitous Lambert Function and its Classes in Sciences and Engineering. In: Pardalos P., Rassias T. (eds) Contributions in Mathematics and Engineering. Springer, Cham

Andrea Gulino is a Ph.D. candidate at the Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB) of Politecnico di Milano, Milan, Italy. He received his master's degree with honor in Computer Science and Engineering from Politecnico di Milano in 2017. His research interests include cloud computing, distributed systems, systems architecture and big data processing.

Abdulrahman Kaitoua is a researcher in the German Research Center for Artificial Intelligence (DFKI). He received his Masters in Electrical and Computer Engineering from the American University of Beirut (AUB), Lebanon, in 2013, and his Ph.D. with honor in Information Technology from Politecnico di Milano in 2017. His thesis was awarded by the Charafas Foundation. His research interests include bioinformatics, data bases, data mining, and big data processing.

Stefano Ceri is Professor at the Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB) of Politecnico di Milano. His research has been generally concerned with extending database technology; he has authored over 300 publications (H-index 74) and 15 books in English. He received two advanced ERC Grants, on Search Computing and on Data-Driven Genomic Computing (GeCo, 2016-2021). He received the ACM-SIGMOD Innovation Award (2013) and is an ACM Fellow.